AD-A202 633

The Rochester Robot

Christopher M. Brown (Editor)

Dana H. Ballard, Timothy G. Becker, Christopher M. Brown,
Roger F. Gans, Nathaniel G. Martin, Thomas J. Olson,
Robert D. Potter, Raymond D. Rimey, David G. Tilley, Steven D. Whitehead

Technical Report 257
August 1988

DTIC
ELECTE
NOV 16 1988
S D
E

UNIVERSITY OF
ROCHESTER
COMPUTER SCIENCE

This document has been approved for public release and sale; its distribution is unlimited.

88 11 16 042

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER TR 257	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) The Rochester Robot		5. TYPE OF REPORT & PERIOD COVERED Technical Report 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Christopher M. Brown (editor)		8. CONTRACT OR GRANT NUMBER(s) DACA76-85-C-0001
9. PERFORMING ORGANIZATION NAME AND ADDRESS Dept. of Computer Science 734 Computer Studies Bldg. Univ. of Rochester, Rochester, NY 14627		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS DARPA / 1400 Wilson Blvd. Arlington, VA 22209		12. REPORT DATE August 1988 13. NUMBER OF PAGES 66
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Office of Naval Research Information Systems Arlington, VA 22217		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)
Distribution of this document is unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
Active Vision
Animate Vision
Robot Vision
Real-time Vision
Kinetic Depth
Vergence
Tracking

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
The Rochester Robot is a unique design that was built to study the use of real-time vision in cognition and movement. The major feature of the robot is the robot head. The head consists of binocular cameras that can be moved at over 400 degrees per second. During rapid movements visual data can be analyzed with a pipeline processor and used to control the motion of a body. The body is a PUMA 761 six degree-of-freedom arm which has a two meter radius workspace and a top speed of about 100 cm/second. These features combine to give the robot the capability of reacting to features in its environment in complicated ways in real time.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

DTIC COPY INSPECTED 6

Accession For
NTIS GRA&I
DTIC TAB
Unannounced
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or Special
A-1

The Rochester Robot

Christopher M. Brown (Editor)

Dana H. Ballard, Timothy G. Becker, Christopher M. Brown,
Roger F. Gans, Nathaniel G. Martin, Thomas J. Olson,
Robert D. Potter, Raymond D. Rimey, David G. Tilley, Steven D. Whitehead

August 1988

Acknowledgements

This research was funded by the DARPA U.S. Army Engineering Topographic Laboratories Grant DACA76-85-C-0001, the National Science Foundation Grant DCR-8405720, the National Science Foundation Coordinated Experimental Research Grant DCR-8320136, the National Institutes of Health Grant NS22407, and the Air Force Systems Command (Rome Air Development Center, Griffiss AFB, NY, 13441-5700) and Air Force Office of Scientific Research (Bolling AFB, DC, 20332) Contract F30602-85-C-0008, which supports the Northeast AI Consortium. Edmond Lee made contributions to the blob extraction software and rewrote the homogeneous transform library. Peg Meeker did conscientious text processing. The Xerox Corporation University Grants Program provided the equipment used in preparing the paper.

1. Background and Overview

This report describes the background, current state, and future directions of active vision research at the University of Rochester. At Rochester a broad and deep program of research in parallel systems has led to a number of related research foci and collaborative work. In this report the collaborations with the Mechanical Engineering and Electrical Engineering Departments are most directly relevant, but our longstanding joint associations with the Center for Visual Science have an important bearing on the form and content of much of our work. Within the Computer Science Department, we are collaborating actively with the systems research group in their development of new paradigms and methods for parallel programming environments for large-scale MIMD computers [LeBlanc et al. 1988, Brown 1988, Fowler et al. 1988].

The central focus of vision research at Rochester for some time has been its relation to acting, behaving systems [Aloimonos et al. 1987, Bandopadhay 1986]. Recently we have acquired and commissioned our major robotic hardware, and integrated it with our existing vision and parallel computing facilities. Also we have recently begun serious work on the higher, cognitive levels of active robotic behavior, the planning levels. These recent developments have allowed us to

broaden our scope from *active vision* to *active intelligence*, and to pursue the goal of an integrated intelligent robotic system.

The next section defines our research objectives in intelligent robotics. Section 3 briefly describes the components and data paths of the Robotics and Vision Laboratory. Then follow four sections that describe our four major research directions: head and body control (Section 4), animate vision (Section 5), cognitive control (Section 6), and neural networks (Section 7). Section 8 lists some basic utility and pilot research projects. Two appendices outline related projects: Appendix 1 describes a design for a programming environment for our real-time image processing system, and Appendix 2 describes new research on foundations for control of highly flexible robots.

The following is a guide to the researchers and authors of technical sections, whom the editor thanks sincerely. Dana Ballard wrote Sections 2 and 7. Rob Potter used the cepstral filter for vergence, and Tom Olson implemented the fast cepstral filter. Tim Becker built the Sun-Puma interface. Dana Ballard and Altan Ozcandarli did the kinetic depth system. Chris Brown and Ray Rimey did the kinematics and stereo work. Dave Tilley implemented the adaptive tracker and is designing the MaxVideo programming abstractions. Roger Gans of Mechanical Engineering did the work on flexible robots. Steve Whitehead is pursuing the data-flow planning research. Nat Martin is writing ARMTRAK. Ray Rimey implemented and integrated BMW.

2. Active Intelligence

2.1. The Idea and the Technology

The goal of building intelligent robots has had a long intellectual history, but was firmly established as a scientific discipline with the advent of artificial intelligence in the 1950s. The earliest autonomous attempts at robots were made in the ensuing decade. The principal lesson of those experiments was a reappreciation of the difficulty of the tasks and goals. Since that time research has been largely theoretical, with formal efforts in the subdisciplines of knowledge representation and planning. One reason for the emphasis on theoretical work at the expense of experiments was that it was appreciated that fundamental advances in hardware capability would be needed both in raw processing power and in parallel computer architectures to achieve real-time performance.

Recently, there have been a number of technical advances on the hardware front. These include: (1) the development of scalable parallel computer architectures, (2) the development of parallel pipeline computers, (3) the development of minaturized sensors such as CCD cameras, (3) the refinement of software for controlling kinematic chains. These developments have made it attractive to integrate previous theoretical advances with a new push in experimental research. While simulation is an important tool in the development of experimental systems, the real systems are sufficiently complex that important issues can only be settled by building the real thing. Another important point is that as intelligent robots become more complex, the cost of building the real system begins to compete successfully with the cost of simulation.

At Rochester the Department of Computer Science has committed to a long-term research program to integrate theoretical and experimental results toward the unified goal of developing intelligent robots. This decision requires advances in artificial intelligence, systems, and theory. The experimental focus of our program centers around two computer systems: the Rochester Robot and the BBN Butterfly computer.

The Rochester Robot is a unique design that was built to study the use of real-time vision in cognition and movement. The major feature of the robot is the robot head. The head consists of binocular cameras that can be moved at over 400 degrees per second. During rapid movements visual data can be analyzed with a pipeline processor and used to control the motion of a body. The body is a PUMA761 six degree-of-freedom arm which has a two meter radius workspace and a top speed of about 100 cm/second. These features combine to give the robot the capability of reacting to features in its environment in complicated ways in real time.

The Butterfly is a BBN computer architecture based on processor intercommunication over a permutation network that allows large scale parallelism. The current largest machine that we have has 128 processors, and the most advanced is a 24-node Butterfly Plus Processor upon which the current systems research is being performed [Scott et al. 1988]. Extensive experimentation [Brown 1988] has shown that for algorithms that can exploit the parallelism of the machine speedups of a factor of a hundred can be achieved rather easily. The attraction of the Butterfly from the point of view of Robot design is first that it allows a workbench with which to experiment with highly nonlinear control protocols and second that the parallelism offers the hope of achieving the goal of real-time responses. Building a system requires extensive organization into coherent research directions. We have organized the robot project around four research tasks.

2.2. Research Tasks

The Development of a Real-Time Head Body Control System. The fundamental problem an animate vision system has to contend with is that its motion produces effects that are on the order of the time constants of its photodetectors so that its vision system must be degraded by motion blur.

Increasing the sensitivity of the detectors is not helpful as the sensors also have to operate in large dynamic ranges. Primate vision systems have evolved to have sophisticated control systems that allow them to fixate on points in the world and change gaze to selected targets. We believe that in order to perform properly, robotic systems must duplicate this functionality. To that end we are developing a head/body control system that has four components.

1) A reflex system that changes binocular gaze with body movements so as to maintain the point of fixation. This is a straightforward transformation to compute the changes in gaze coordinates given the change in neck coordinates. The neck is defined to be the wrist coordinate system of the PUMA and the gaze coordinates are the three angles of the head control system.

2) A vergence system that controls the binocular coordination of the two cameras. The control system is eye dominant. The non-dominant camera can be moved to aim at the point in space that is defined by the first intersection of the optical axis of the dominant camera with solid material.

3) A saccadic system that controls the change of gaze of the eyes with respect to the head. Targets selected by some externally defined computation can be fixated with high speed motor movements.

4) A pursuit system that allows the fixation of moving (with respect to the world) targets. The pursuit system can also be used to assist in maintaining a fixed gaze (see 1).

Animate Vision. With the advent of animate vision systems, the computational theories of vision have to be reworked. This has not been appreciated in vision because self motion has been mathematically equated with environmental motion. While in an abstract sense this assumption was justified, in a practical sense there is a huge difference between self motion and world motion for two reasons. First of all, fixating systems have nonhomogeneous resolution owing to motion blur. Second, animate systems can control their motion with respect to the environment. These two considerations combine to allow animate computational algorithms that are both different and simpler than passive-camera algorithms.

Planning and Multi-Tasking in Complex Dynamic Environments Animate vision is directed towards the computation of data needed in single tasks but any autonomous system must be capable of managing many different tasks of dynamically-varying priorities. To this end hierarchical control systems are needed that must be tuned to the environmental load.

Neural Network Representations. Neural network representations are a relatively new development that provide answers to two problems that have plagued researchers in robotics. First, neural network representations can be learned. This means that efficient representations can be chosen from examples rather than selected a priori. Second, neural network representations use shared representations. This means that the same hardware is used to encode several different relations economically. The result is that the representation may be able to generalize correctly given new samples or conditions.

3. Robot System Overview

3.1. Major Components

Figure 1 shows the layout of the University of Rochester Robotics Laboratory. The "robot head" (Figure 2), built as a joint project with the University's Mechanical Engineering Department, has three motors and two CCD high-resolution television cameras providing input to a MaxVideo® digitizer and pipelined image-processing system. One motor controls pitch or altitude of the two-eye platform, and separate motors control each camera's yaw or azimuth, providing independent "vergence" control. The motors have a resolution of 2,500 positions per revolution and a maximum speed of 400 degrees/second. The controllers allow sophisticated velocity

and position commands and data readback. The robot arm has a workspace with a two meter radius, and a top speed of about a meter/second.

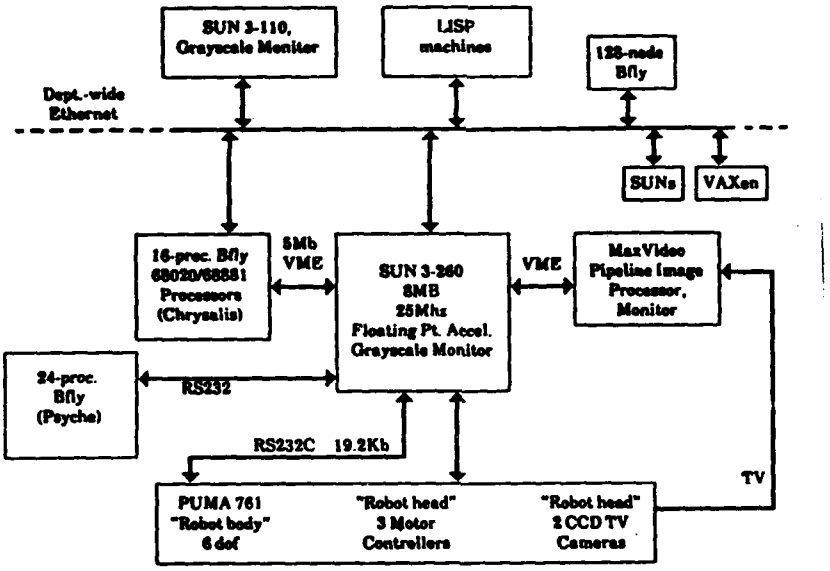


Figure 1: The University of Rochester Robotics and Vision Laboratory.

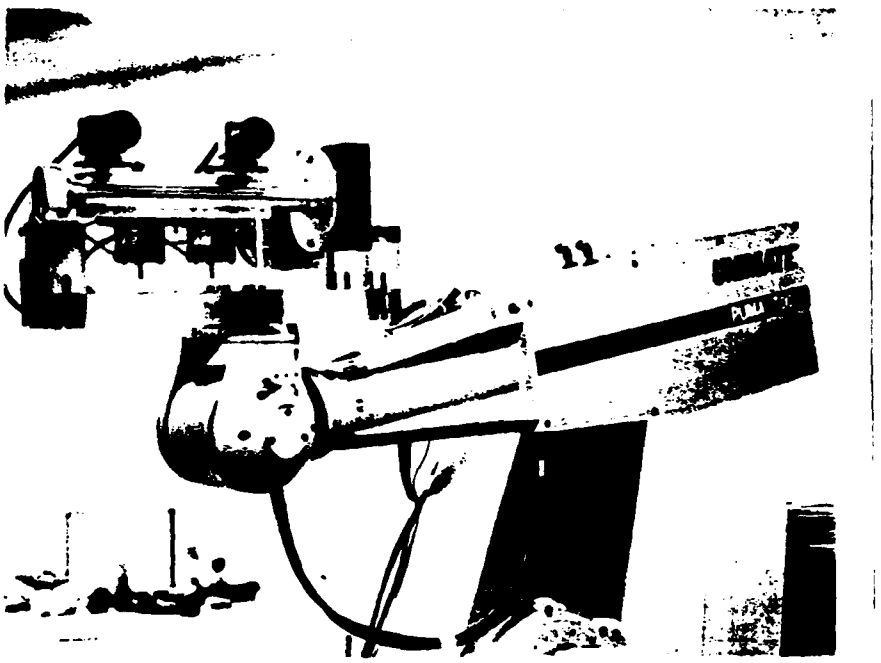


Figure 2: Rochester's robot head has two cameras, a common altitude control, and independent azimuth controls. It is mounted on a six-degree-of-freedom manipulator.

The MaxVideo system consists of several independent boards that can be cabled together to achieve many frame-rate image analysis capabilities (for example, frame-rate depth map calculation from optic flow generated by active head motion [Ballard and Ozcandarli 1988]. The ROIStore frame buffer is one of the most complicated special-purpose boards, allowing storage of multiple images and manipulation and transmission of arbitrary subimages. Other boards allow 8×8 or larger convolution, arbitrary signal processing computations, pixel-wise image processing, cross-bar image pipeline switching for dynamic reconfiguration of the image pipeline, look-up tables, histogramming and feature location. More details are given in Appendix 1 and Section 3.2.

The Puma robot (Figure 2) is presently controlled by a dedicated LSI-11 computer implementing the proprietary VAL execution monitor and programming interface. Our own software (Section 3.2) will exercise this interface to the fullest, and if it should prove inadequate we shall investigate special interface boards and software, such as RCCL [Hayward 1984, Paul et al. 1984].

There are some forty Sun computers and twelve LISP machines in the Department. The Butterfly Parallel Processors in the lab are described in more detail in Section 3.2. They offer flexible parallel processing, which may proceed in a SIMD or MIMD style, and a choice of vendor-supplied or locally-produced programming and debugging environments. The Butterfly Plus configuration can run the Mach system [Accetta et al. 1986] and will be the target architecture for most local parallel programming environment and operating system development work.

3.2. Communication between Sensing, Cognitive, Control, and Effecting Systems

High-bandwidth communication is necessary between computing, sensing, and effecting elements in an active vision system. The current laboratory is adequate in some aspects, and we are contemplating improvements where necessary. The system consists of the data paths shown in Figure 1. This section describes the hardware and software interfaces in the system.

The MaxVideo software system runs on the host (a Sun 3/260). The host has a VME bus interface. A VME bus repeater is attached to the host at one end and to a VME chassis at the other end. This chassis contains our selection of MaxVideo image processing boards, which currently include the following.

1) one Image Digitization D/A, A/D Board (DigiMax)
2) three Region Of Interest Image Memory Boards (ROIStore)
3) two Real-Time Image Convolution Boards (VF-MKII)
4) two Signal Processing Boards (MAX-SP)
5) one Real-Time Feature Extraction and Histogram Board (Feature MAX)
6) one Single-Board Signal Processing Computer (Euclid)

We are planning to expand this board set substantially, which will require another chassis. These boards are all register programmable and are controlled by

the host via the VME bus. This interface is a 24d16 device and has a throughput of about 2 MB/second. This could be upgraded to a 24d32 device which would bring the throughput to about 3 MB/second. The user calls MaxVideo routines that read and write data to and from these boards over the VME bus via a device driver that maintains a queue of register operations. Images are transferred between these boards via high speed image busses on ribbon cables. The Euclid board has the advantage of having both the high speed image bus interface and a VME Bus Master interface. Thus this board can program the other boards in the same manner as the host. The MaxVideo device can operate either in continuous mode or interrupt mode. We have not so far successfully used interrupt mode.

The 16-node Floating-Point Platform Butterfly Parallel Processor® (FPP-BPP) has a VMEbus connection that mounts in the same card cage as the MaxVideo and motor controller boards. The interface supports interrupts and data communication. No significant calculations are now performed on the BPP, but Chrysalis, its native operating system, can support real-time applications, and there are several programming environments available [LeBlanc et al. 1988]. Not only could the BPP implement higher cognitive functions, the bandwidth of the VMEbus allows it to do motor control.

Our parallel systems and parallel vision research lines converge at the higher end of the computational spectrum, with the MIMD computers that have occupied us for several years [LeBlanc et al. 1988]. A branch of our current work in parallel systems is devoted to implementing the Psyche operating system, which features a number of novel ideas [Scott et al. 1988]. Psyche is being built on the *24- node Butterfly Plus Processor®* (B+P), which makes more sophisticated use of the M68020 hardware (in particular, memory management) than does the FPP-BPP. Psyche's first application environment will be the Robotics Laboratory, and we will be working closely with the systems research and development staff to assure that the B+P can be a useful tool for high-level vision and robotics computation.

To keep the initial implementation of Psyche as simple as possible, a serial line interface (as opposed, say, to a VMEbus interface) is the first communication channel between the Robotics Lab and the B+P. This relatively slow link will, for instance, allow access and updating of slowly-changing or environmental descriptions, such as the location of objects in the environment. Faster, and perhaps multiple, channels such as VMEbus will be added as programming resources permit. Ultimately the high internal bandwidth of the B+P makes it a promising architecture for the cooperating modules that we believe characterize the higher levels of an intelligent autonomous system.

The communications link between the Sun workstation and the Puma robot controller consists of multiple layers [Unimation 86b]. The physical interface between the workstation and robot controller is an RS232C cable running between the serial port on the workstation and the supervisor function port on the robot controller [Unimation 1986a]. This connection runs at 9600 baud.

The link control layer uses a subset of DDCMP (Digital Data Communications Message Protocol) [DEC 1978, Unimation 1986b]. This protocol provides a full-duplex error free connection between the higher level communications layers. On

the Sun, DDCMP is implemented as a special pseudo-device written as a tty line discipline in concert with a user level library. The tty line discipline reads and writes ddcmp packets and the library implements connection start up, packet acknowledgments, and application code reads and writes. This software was developed and distributed by the Electrical Engineering Department at Purdue University.

The next communication layer provides logical communication channels. This allows simultaneous conversations between multiple Val II tasks. (Val II is the native Puma control system provided by the vendor. It runs on an LSI 11/73 CPU.) Each of six tasks communicates over a separate logical channel. Some of the tasks are:

1) Monitor commands and messages.
2) User program input and output directed to the terminal.
3) Disk commands and data transfers.
4) Terse robot status information.

The software that implements these tasks on the robot controller is part of Val II from Unimation. The Sun workstation side of this interface was inspired by the analogous interface developed at Purdue, but has been completely reengineered here to provide more flexibility and speed. The software package provides C language routines to:

1) Get the current robot location in terms of standard coordinates or joint angles.
2) Move the robot to a specified location in terms of standard coordinates or joint angles.
3) Set the speed of the robot.
4) Set the location and orientation of the tool tip.

The software is organized as a UNIX C language library. The routines described above can be called from the application's program running under UNIX.

In addition to parallel computing, *distributed computing* will be important particularly when using vision routines for high level cognitive tasks. Distributed computing will make use of the Department's Ethernet network. The bandwidth of this network is more than adequate for the distributed computing foreseen for the near future.

The demands for distributed computing will be driven primarily by those using vision routines in investigating high level cognitive tasks. Because these tasks involve relatively short symbolic messages, the bandwidth of the network is less problematic than it is for those routines which manipulate images. One important consideration is the substantial software base written in Lisp. Therefore it is important to provide *communication with the departmental Lisp machines* including both the Symbolics and the Explorers.

The routines supporting distributed computing are based on the Sun workstations running UNIX. The server running on the workstation generates a set of stream based sockets with which processes on other machines on the network can

communicate. Though stream based socket limit the number of clients the server can serve, the simplicity of communicating with the Lisp machines makes streams an attractive choice. Common Lisp, the primary dialect of the department, has communication protocols based on streams. By maintaining stream protocols on the Suns, difficulty of communication is minimized. Though communication by streams limits the number of clients it is unlikely that more than twenty servers would require the services of the vision lab at any one time. Moreover, it is very unlikely that the lab has sufficient resources to satisfy such a large number of clients.

4. Real-Time Head and Body Control System

One major focus of our work is the integrated, hierarchical, reflexive control of the sensory and motor systems of the robot [Ballard 1987b]. The low-level head and body control serves to implement commonly useful relatively complex actions that arise in response to the environment, or can be initiated and then automatically maintained (such as object tracking). The subsystems map relatively cleanly onto known animate control systems and capabilities. They divide into three major systems.

1) Low-level reflexes or capabilities such as gaze direction, touching, saccades, binocular stereo, VOR-like compensating body and head movements.
2) The smooth pursuit or tracking system.
3) The vergence system.

4.1. Reflexes and Low-Level Capabilities

Gaze Direction Control. We now understand the coordinate systems and state descriptors used by the VAL programming environment for the Puma arm, have made first steps toward calibrating the cameras for active vision applications, and have derived several useful relationships between state descriptors of the system used by the robot control language VAL and by vision researchers [Brown and Rimey 1988]. Besides the conversions, we now support various basic capabilities, such as the ability to direct the gaze to a three-dimensional location. Inverse kinematics translates descriptions of positions in world coordinates to state descriptions of a manipulator in that position. The calculations for the robot are provided by VAL. The calculations for the head supplement those for the robot, and tell how to relate desired camera positions and orientations both to robot commands and camera motor commands.

Touching. A probe, or proboscis, may be rigidly attached to the head. By using the VAL command to define the TOOL coordinate system [Brown and Rimey 1988] at the tip of the probe, the probe can easily be commanded to explore and interact with a known 3-D environment. We use the capability to verify that 3-D locations derived from stereo are accurate, but other uses are possible as well.

A more complex task is to interact with a dynamic world in a task involving both sensing and acting. One such task is to keep a balloon in the air with a paddle attached to joint six of the robot, along with the head. This task requires object acquisition and tracking, and coordinated head and eye movements. It also requires constructing models of three-dimensional motion (balloon trajectories) and

accomplishing manipulation goals (bouncing the balloon), which of necessity affect the location of the head. An application of this complexity calls for considerable flexibility in resource allocation (Section 6), and raises issues of the interaction of reflexes with each other and with commands generated by higher cognitive processes. Thus it is a suitable application for which to develop novel scheduling and control algorithms.

Image-Based Saccade Control. Camera calibration measurements and the small angle approximation make it possible, with reasonable accuracy, to direct the gaze of the camera so that a pixel at initial image location (x,y) moves, in the redirected gaze, to image location $(x+dx, y+dy)$. The model of the calculation is simply that the camera pivots around the front principal point of its lens, that the camera is adequately modeled as a pure point-projection system (pinhole), and that the small-angle approximation ($\tan(x) = x$) applies. None of these things is strictly true. The worst failure of the approximations is the first assumption. In the robot head, the cameras pivot around axes that are distant from the lens principal point by some 35mm in the azimuth (image x) direction and 65mm in the altitude (image y) direction.

Three-Dimensional Saccade Control. The eyes may be accurately directed to look at a known point in space. Thus if 3-D point locations are known (derived from stereo, prior knowledge, manipulation, or other means), saccade control can be free of approximations. Providing this capability amounts to solving an inverse kinematic problem. We have the capability to derive two camera-control angles for altitude and azimuth from the (x,y,z) position of the point to be fixated on the optic axis and the current state of the robot body (the location of the head in space) [Brown and Rimey 1988].

Binocular Stereo. Camera calibration by definition allows the association of any image point with a three-dimensional locus (in our imaging model, a line of sight) that contains the location of the scene point corresponding to the image point. Thus two calibrated images from different locations containing the same scene point allow it to be fixed in space as the intersection of the lines of sight. This observation motivates a binocular stereo algorithm we have implemented [Duda and Hart 1973]. Since our model of the imaging system is linear, the $\hat{x}$ or $\hat{y}$ image position of a scene point is a linear function of its spatial (x,y,z) position and the camera parameters. Thus more than two $\hat{x}$ or $\hat{y}$ coordinates from two or more images derived from known camera models (arbitrary position and orientation) yield an overdetermined system of equations in spatial (x,y,z), which may be solved by pseudoinverse techniques. We also have implemented this method, and are exploring its properties. At present the two methods provide comparable results, but each has advantages and disadvantages. The stereo calculations assume the correspondence problem is solved: we are using the cepstral filter to obtain correspondence in the related application of vergence (Sections 4.3 and 4.4).

The binocular stereo algorithms and eye motor control and readback commands can be integrated to derive the 3-D location of an object visible in both cameras, regardless of their altitude or azimuth settings. The Ball Mapping Workbench (Section 5) currently assumes the cameras are at zero altitude and azimuth. Improving this capability allows the active vision system to relate its sensors to the

three dimensional world in full generality, and is necessary for further serious work in eye movements.

Compensating Body and Head Movements. The visual system can be used to induce an object-centered coordinate system useful for manipulation and recognition [Ballard and Ozcandarli 1988]. The user-definable TOOL coordinate system capability provided by VAL gives remote object-centered coordinate systems as well, which can be used in the following manner. Define the TOOL coordinate system at a location some distance D along the optic axis of a camera. To foveate the object (center it in view) and put it at the distance D from the camera, simply move the TOOL to the object. To view the object from different angles while keeping it foveated, command the robot to move to locations that have different orientation components of the TOOL location description and the same position components. The effect is to pivot the head around the (remote) TOOL origin. This capability to fixate a point in space while moving the head achieves similar goals to the VOR reflex, except it involves only movements of the head and its supporting body, and does not include movements of the eye motors. Further, if the eyes are verged at the distance D then foveation in both eyes will be preserved by the reorienting motions. This idea leads directly to dynamic segmentation (Section 5.2).

There is much future work to be done in the creation, and especially in the integration, of low-level reflexive behavior. Head-eye coordination is a basic cabability of active vision. *Integrating saccade-control with object location capabilities* and robot motion commands is a discrete approximation to smooth hand-eye control, which involves synchronized head and eye movements and understanding differential motion relationships.

Let us call "head motions" (or "robot motions") those which the body effects to move the head in space, and "eye movements" those which the head effects to move the cameras in altitude and azimuth. A logical next step is to close the control loop between the motion of an object in 3-D and the control of the robot eyes. This involves integrating saccade-control with object location capabilities and robot motion commands. The work can start with discrete motions, but ultimately should involve synchronized head and eye movements and understanding their differential motion relationships. Many larger projects need this basic capability, and it is important that we understand its performance potential on our system. The work can proceed either on the basis of monocular image coordinates (pure retinal slip), using 2-D saccade control, or by integrating the results of binocular stereo with the inverse kinematics calculations that derive camera control from 3-D object location (3-D tracking). In either case the necessary tight coupling between sensing and moving can be implemented since the eye-motor controllers are accessed by processes executing in the Sun. Steve Whitehead has attained four saccades per second, and Ray Rimey has demonstrated real-time location of nine blobs in a single image. The indications are that quite good head-eye coordination performance should be attainable.

The VOR is a basic reflex that keeps the eyes fixated on a point in spite of head motions. In animals, the VOR is mediated by input from the vestibular canals, as opposed to visual feedback like retinal slip, although the latter can clearly be used to control eye movements in robots and animals (for smooth pursuit of moving objects

when the head does not move), and clearly must interact with the VOR. Our current compensating head and body movements, which maintain fixation at a 3-D point despite head translation in 3-D, are implemented only with Puma joints, and do not involve "eye" or camera control. Thus one obvious goal is to *implement a VOR-like reflex that uses eye movements (camera motors) to stabilize cameras across (six-degree-of-freedom) robot head movements.* This reflex could then be integrated into complex, cognitively-driven head and eye tasks. Ultimately the VOR should be described and implemented in terms of continuous change, requiring the differential Jacobian relationships between body and head. Initially, however, a discrete version is well within the realm of possibility and is a reasonable intermediate goal. The discrete version does not need differential relationships, but simply integrates the the known capability to saccade to a given scene point from any robot state.

The next task is the important inverse problem of coordination between body, head, and eyes when eye movements are driven from the world (tracking a moving object, say). A sample problem is keeping an object in view when the eyes reach their maximum excursion in altitude or azimuth. Discrete-movement body-head-eye coordination extends coordination to include head motions, and requires the inverse kinematics so far developed and robot control from the Sun. The new technical problem introduced here is synchronizing the movements produced by two independent effector systems. If head and eye movements can be adequately synchronized, then Jacobian calculations describing the relationship of differential body and head movements can be used for smooth-motion body-head-eye coordination. This general capability is directly useful in several specific contexts. In the current configuration the control of the robot is indirect, implemented via interprocess communication between the commanding system and the VAL system, which then issues control signals to the Puma. Academic researchers and those interested in real-time systems have usually found this arrangement inadequate [Andersson 1988, Paul et al. 1984, Hayward 1984], and we are exploring alternatives. In the meantime, we shall explore the characteristics of the existing control loops and try to incorporate head motions as well as eye motions into object-tracking. The eyes could move until they approach physical limits, at which time head movements could be produced to accommodate. The obvious compensations are to rotate the head in an azimuthal degree of freedom (joint 6) if the eye azimuth gets near a limit, and rotation around some axis parallel to the head altitude axis if eye altitude gets near a limit. This latter motion is more interesting, since there is not necessarily a single robot joint that generates the right motion. Alternatively, an estimate of the three-dimensional trajectory of the object could be developed and a continuous head movement initiated to follow the object. In such a case, the head movements are dictated by one error signal (periodically sensed variations from the predicted object trajectory) while the eye movements are controlled by another (retinal slip). Several issues emerge: one is how the desired head motion is chosen from the multitude of possible trajectories [Pellionisz 1985], another is the interaction of control of the two motions [e.g., Miles et al. 1985].

As a potentially important issue, we have noticed in our own work and that of others [Brooks 1987], that inverse robotic and vision problems (like inverse kinematics and camera calibration) are normally at the mercy of the model used to describe the system. We believe that the most useful sort of accuracy can be gained not necessarily by engineering the environment to fit the model [e.g., Shafer 1988],

but by leaving the model enough degrees of freedom to fit reality. We also believe in automatic adjustments, accommodation, or learning as a mechanism for maintaining accuracy. We are exploring the idea of learning some inverse problems with backpropagation or recursive backpropagation neural nets [Jordan 1988, Simard et al. 1988]. If these techniques look practical we may choose to incorporate them along with other approaches to control (Section 6).

The vergence mechanism can be driven by several stimuli. We are investigating one currently based on the autocorrelation-like "cepstral" filter (Sections 4.3 and 4.4). Incidentally, this work provides our first example of significant, multi-computer distributed vision computation: the cepstral is computed on the MaxVideo Euclid board. There are several follow-on projects. The cepstral can extend our work in binocular stereo to tackle the correspondence problem using sophisticated features or image subsets, such as those extracted from the image itself in the adaptive tracking paradigm. The cepstral filter provides a way to associate a disparity with an image patch without doing explicit object-recognition or segmentation. Thus its output can be interpreted as a set of "corresponding" pixels between two images. The corresponding sets are the cepstral patch in one image and the result of applying the disparity to the locations of all the pixels of the patch. The stereo geometry calculations could be applied to the pixels of the patch and to its "corresponding" disparity-shifted pixels to derive a 3-D location for the point set. If carried out for a set of patches that cover the image, the calculation yields a coarse depth map [Yeshurun and Schwartz 1987].

4.2. Pursuit System: Adaptive Real-Time Retinal Object Tracking

The MaxVideo pipelined image processor supports frame-rate convolution with any of 128 8×8 templates that have been previously downloaded from the host computer. One straightforward use for such a device is as a flexible feature-detector, such as a line-finder or DOG filter. The output of the convolution board can be sent back to the host vision computer, or can be sent to other MaxVideo boards for processing. One straightforward example is to send the correlation output to the FeatureMax board, which can report the locations of correlation peaks, i.e., correlation values over a threshold, that correpond to feature locations.

The goal of the adaptive tracking project is to implement the more complex capability of extracting the correlation template from the image itself, thus adapting the template in real-time to be a "matched filter" for the desired image pattern. The signal from the correlator may be used to guide eye movements or to implement pattern recognition. We assume that the target can be moving and thus the image of the object may be continually changing by small amounts. We assume that initially the location of the object in retinal coordinates is given, and we take a sub- image at that location. That sub-image is to be used as a correlation template. In order to take into account the evolving nature of the object as it moved, the template is continually updated. Every frame a new subimage is taken. Such an adaptive tracking routine is insensitive to slow variations in orientation of the object or camera head. If however there is a large variation in object orientation, the system can lose track of the pattern. Re-aquisition of such a lost object is a "higher level" problem to be solved as needed by the planning layers of the system. In designing the fixation system for specific hardware, several issues are of interest.

4.2.1. Issues

The first issue is that the VFIR-MKII does convolutions of templates against images that are considered to be streams of pixels. The board has no concept of interlaced frames like those used for RS-170 displays and cameras. Since convolution (and thus correlation) is a neighborhood operator there is a problem if all of the even scan lines come, followed by the odd scan lines. In other words, "there goes the neighborhood." So it is necessary to find a way to "de-interlace" the image frames so that the scan lines pass through the convolver in spatial order.

Another issue is that the result of an auto-correlation is an image in which bright peaks are supposed to correspond to areas that match the template. Unfortunately the resultant image is also affected by the general brightness of the pixels as well as their "matching" quotient, since VFIR implements linear, not normalized, correlation. Thus if there are areas of extreme brightness in the image then these areas are likely to show up as bright or even brighter than the areas that actually matched through correlation. Thus some scheme to normalize the input is deemed necessary.

Another key issue is image and template scaling, so that the resulting image is within a desired range determined by the hardware and the application. If the values are too high then they wrap around so that peaks are seen as very low valleys. If they are too low then they may be indistinguishable.

Another issue is the size of the sub-image. Is an 8×8 template large enough? Does it provide enough context to enable us to track objects of interest? The VFIR-MKII at any time can select one of 128 8×8 convolution kernels. If larger kernels are required then more boards in cascade are needed or the image must be recycled through the VFIR-MKII several times to effect a larger template. More hardware of course costs more money, and recycling the image takes more time. A "best" resolution can be chosen for any particular scene, but the real question is how large a kernel is needed to provide adequate information content.

4.2.2. Strategy

The first issue is to find a way to de-interlace the image. That is, make the temporal order of the scan lines the same as the spatial order. There are at least two ways of accomplishing this task. The first is to write the image into memory field by field. After two field times have elapsed (one frame time), the entire frame is in memory and in spatial order. The frame can then be read out of memory in a "non interlaced mode" and processed as such. This approach requires double buffering of the images to work at frame rates, which in turn requires an additional memory buffer (ROIStore) board.

The second option (the one we adopted) is to reduce the image size by ignoring the second field of the frame. Essentially, this is subsampling in the vertical direction by a factor of two. In order to have the spatial resolution equal in both directions we decided to subsample in the horizontal direction as well. This resulted in one 256×241 image every 30th of a second. But since one field of the image is essentially

thrown away, there is no need to double buffer because one field time can be used for writing and the other for reading. As hardware permits we may eventually implement the double buffering scheme.

The second issue is that of the unnormalized, linear correlation, which responds to brightness as well as pattern. Bright regions in the image they may very well end up brighter then the auto-correlation peak. A preprocessing method is being used to help fight this problem. Filtering the image with a Laplacean removes low frequency bright spots from the image.

The third issue is that of scaling. We are working with signed images such that black is -128 and white is 127. With this choice we need to scale the image and correlation template such that the resulting image is within that range, with the correlation peak as close as possible to 127. We are currently using an ad hoc method that assumes that the highest peak detected in the image is the autocorrelation peak of the template (the match of the template with the area that the template came from). Thus the sum of the squares of the elements of the template is the predicted highest value in the image. The VF-MKII board automatically shifts the each resulting pixel value by six bits (to the right), to prevent overflow. So shifting the sum of squares by six bits produces the highest value in the resulting image, and allows scaling the template to make the peak within the desired range.

The final issue is that of template size. As of now it is unresolved: we are using an 8×8 template at different image resolutions. Our future decisions will be dicatated by experience.

In debugging the package we work with static synthetic images exhibiting features with known and easily-checked autocorrelational properties. They are composed of geometric shapes (ellipses, rectangles, etc.) on a variety of backgrounds, some with low-frequency intensity variation. The images are de-interlaced, run through a filter to reduce irrelevant brightness variation, and then correlated against a scaled mask selected from this filtered image (Figure 3). Applying the setup to static real-world images gives the expected results, with high-frequency patterns yielding strong peaks, and the possibility of other strong peaks from related patterns. The next step is to interface the tracker output to the feature-location board FeatureMax, whose output is sent to motor controllers to accomplish real-time tracking. The FeatureMax board can threshold and emit (x,y) image locations for above-threshold pixels at frame rates, and the control loop connecting the image analysis system to the camera-control system is quite fast. Several other applications successfully use the hardware in this mode, so we anticipate no technical problems (Figure 4).

4.2.3. Results

The project is still under development. So far the de-interlacing, preprocessing, and correlation aspects have been completed. The work of thresholding the correlation output and moving the cameras is mostly an integration effort. The de-interlacing of the image worked quite well. The subsampling lost some information (down to 256×241), but still left enough resolution for the tracker to perform. The Laplacian preprocessing performed as expected.

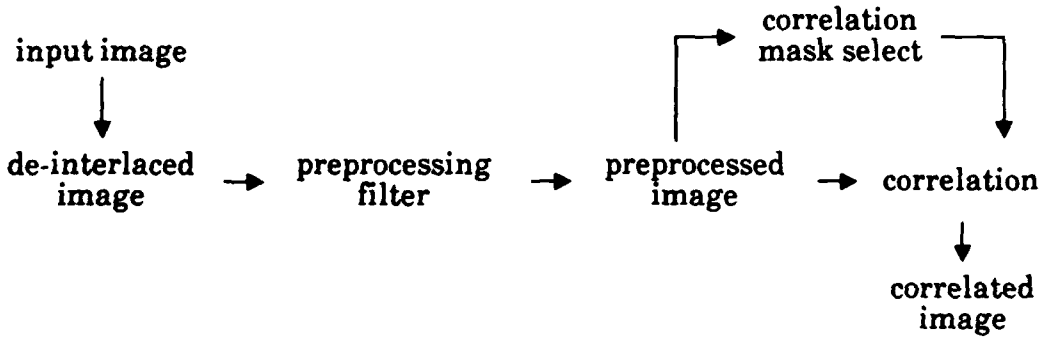


Figure 3: Flow diagram of the adaptive autocorrelation configuration.

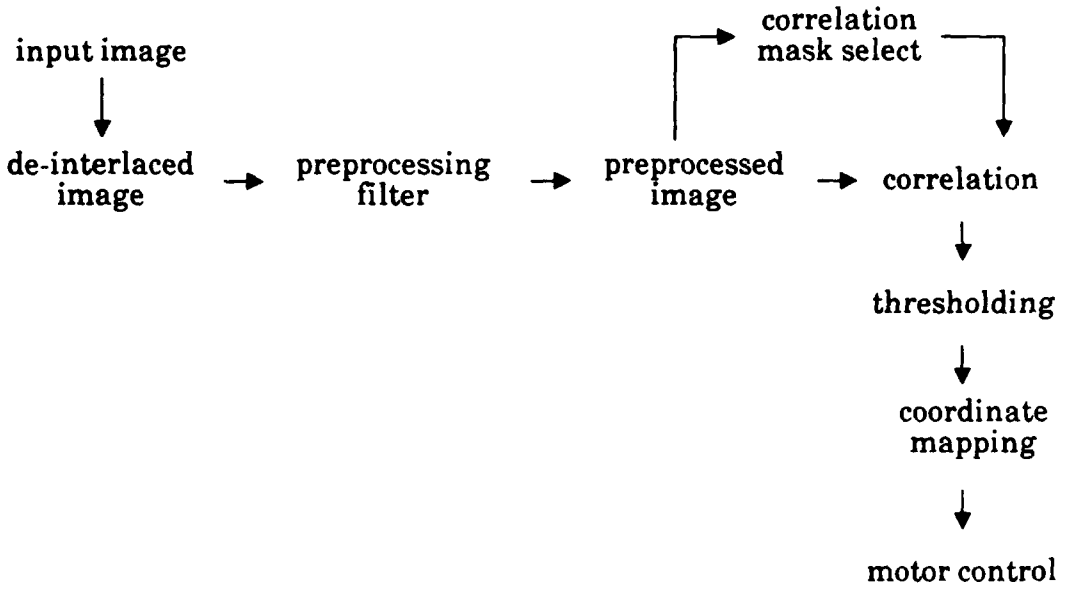


Figure 4: Flow diagram of the adaptive tracker.

The heuristic mask-scaling algorithm went through several stages of evolution. The final variation works for many contrived and real-world images yet does not work for all images. This is an area still under development. The problem seems to be that once the scaling factor has been calculated for scaling the peak to the proper range, it is difficult to figure how to spread this factor among the 64 elements in the correlation template. Low numerical representational accuracy (eight bits) is the basic limitation here. Unfortunately, it is those values most vulnerable to scaling problems (high values that wrap around to low negative values) that are most important for tracking.

The question of template size is the least resolved issue in the system. It seems that an 8×8 template at the current resolution may be too small for general use, given the high resolution the digitizer is capable of producing. Digitizing at a coarser resolution improved performance. This issue has not been studied thoroughly enough to make any final conclusions, but empirically subsampling by a factor of two to 8 seems to yield acceptable results. Figure 5 shows the performance of the correlator on a static scene.

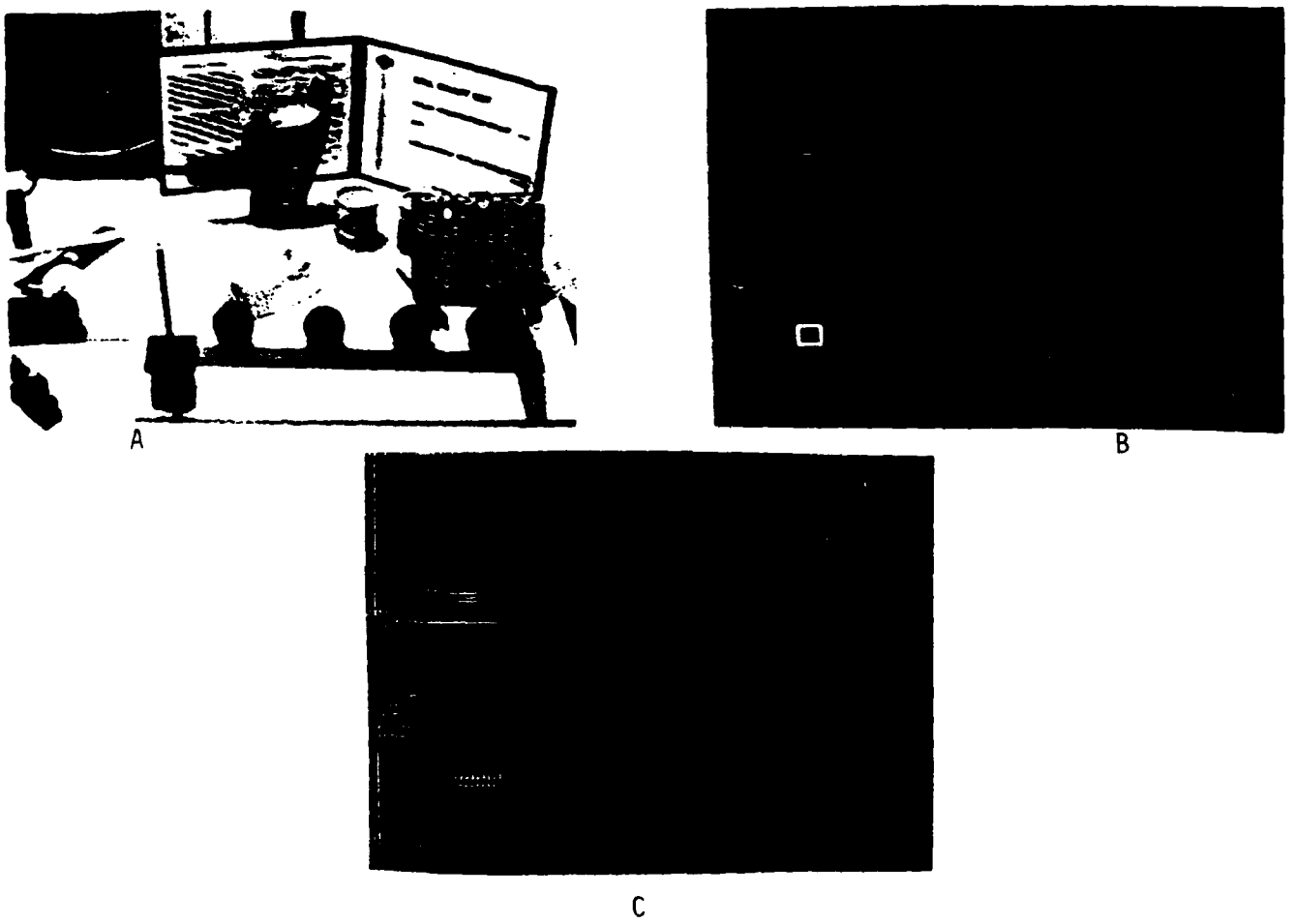


Figure 5: (A) Digitized input scene. (B) Laplacian preprocessing to lower contrast and enhance high frequencies, and rectangle showing location from which template is taken. The pattern is part of a railroad track. (C) Output of correlator, showing anticipated periodic response to periodic track, and potential false peaks from strong edges in the scene.

4.2.4. Future Work

Certain details remain to be addressed in the existing system: we must *integrate the working components* and *resolve the auto-scaling issue*. The goal is to make the fixation system a finished building block that can be used as a component in future research efforts. A natural follow-on project would be to *acquire and then track an*

object, thus integrating the saccadic and pursuit systems. All the projects that have so far involved real-time tracking have used a very simple conception of "object," amounting to "2-D blob." Initially this simplification was due to limited bandwidth between image storage and computational hardware. More recently it has been due to a reliance on simple defininitions of "feature" to present to the FeatureMax board of MaxVideo. The FeatureMax board outputs the image $(\hat{x},\hat{y})$ location of pixels that are designated as feature pixels. Thus far we have only recognized pixels over a threshold as features.

It is not hard to imagine a complex capability that detects an object by some criterion, extracts a promising area of it to track (by running an interest operator that predicts the autocorrelation properties of the area), and then locks on to the area by iteratively extracting a correlation mask, performing the correlation with the VFIR-II board, sending the correlation to FeatureMax, processing the output by peak-detection possibly combined with prediction to compensate for time delays, and producing output that can be used monocularly with 2-D saccade control to track an object in image space or binocularly with 3-D saccade control to track an object in 3-D space.

The same control complexity without sophisticated use of the VFIR board would be required of a system that simply extracts above-threshold pixels with FeatureMax, groups them into regions, and *characterizes the region shape properties* by scan-line techniques. This version of the problem tracks high-contrast objects of distinctive shape, and does the shape pattern recognition in a general computer rather than in a special board. This application is complex enough to admit both traditional and novel approaches to control strategy (see Section 6).

4.3. Vergence with the Cepstral Filter

4.3.1. Gross Vergence

Binocular stereopsis requires that two cameras be looking at the same object at the same time. If the cameras have a fovea or a limited field of view, then they must be verged to bring the object of interest into view in both cameras (Figures 6 and 7).

We have been working with vergence primarily in the context of saccades. When the gaze is shifted from one object to another, the vergence angle needs to be adjusted because the new object may be at a different distance than the old one. If the distance of the new object of interest is unknown at the time of the saccade then the correct vergence angle cannot be determined at the time of the saccade. The images from the left and right cameras must be compared to determine the vergence error. The human psychophysical literature divides human vergence into gross or "disparity" and fine or "fusional" mechanisms [Miles 1985]; perhaps it is the disparity vergence mechanism that is responsible for this post-saccade verging.

The most straightforward way to do post-saccade verging would be to find the object of interest in the images from the left and right cameras, and use these locations to determine the amount each camera should be moved to center the object on the foveas. Unfortunately, this task is very difficult. Object recognition and correspondence determination are of course very hard problems, but they are even

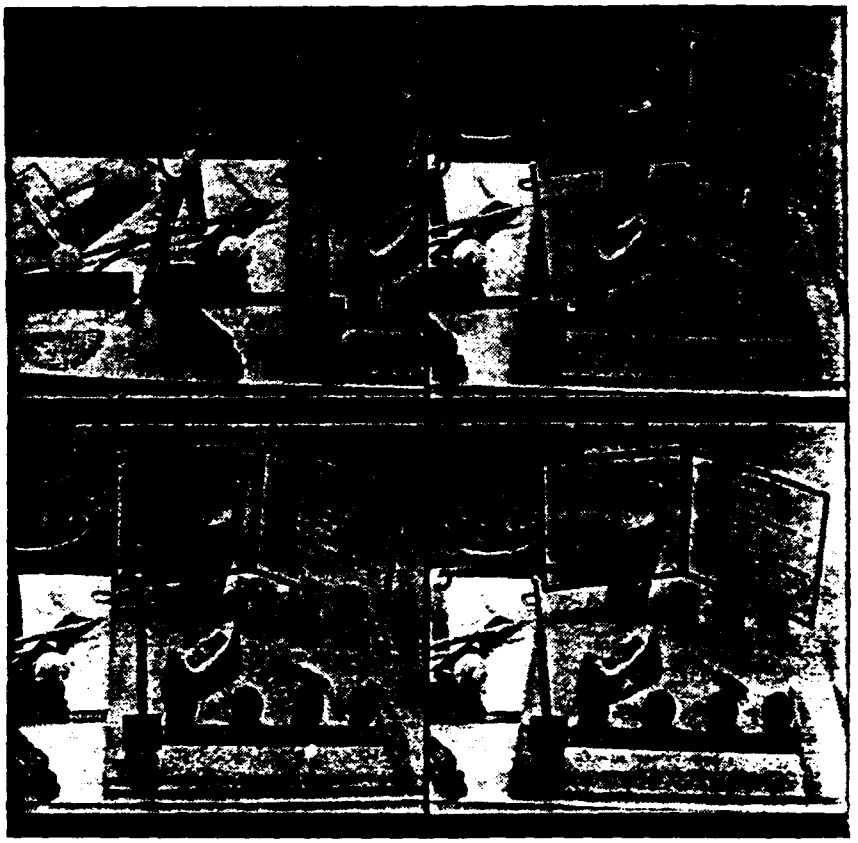


Figure 6: (A) A left/right image pair of some objects on a table, before verging (the originals were 512×512×8 bits). Note that only just over half of the image area is "shared." (B) An image pair of the same scene after the left camera was physically verged by the cepstral algorithm. The drill has essentially no disparity; the only objects with large disparity are those far from the drill in depth, such as the tall object in the foreground and the chair base and floor in the background.

harder just after a saccade. If the saccade is between objects at very different depths, then correspondence determination will be made more difficult because each object will appear in very different places in the two images. Locality assumptions will generally not hold. In the worst case the object of interest may simply not appear in one of the images.

What is needed is a way to get both cameras to share approximately the same field of view, so that most objects are nearly aligned in the two images. By insuring that most objects are in view by both cameras and that their retinal positions are close, we can simplify the task of verging directly on a single object of interest. This gross vergence system must of necessity consider much or all of the field of view of each camera.

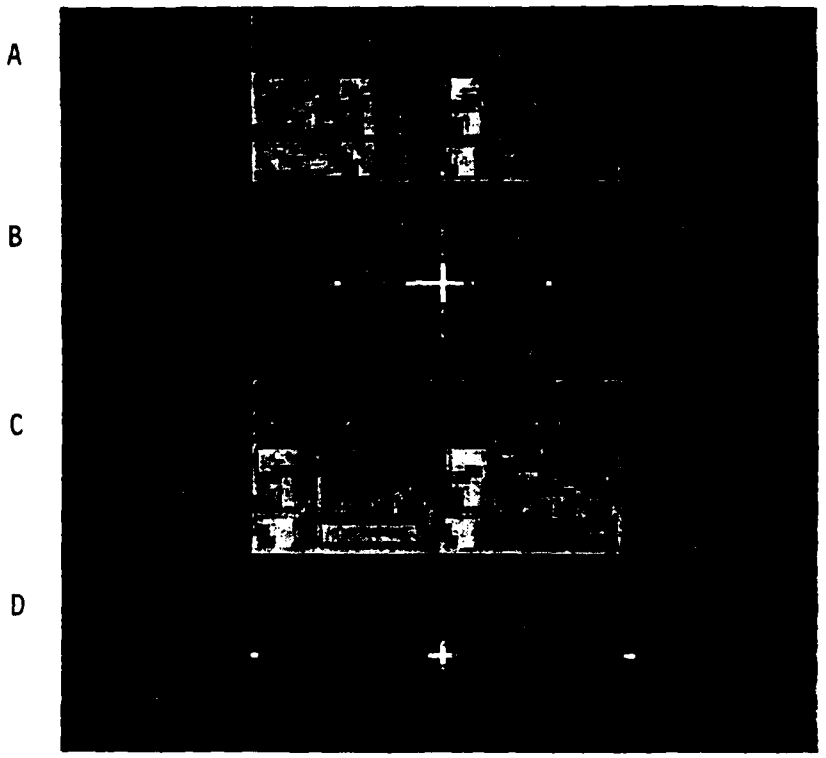


Figure 7: (A) The image pair from Figure 6A, reduced to 32×32×8 bits (by averaging) for computational efficiency. (B) The symmetrical cepstrum of (A). Note the two peaks to the side of the central cross. The position of these peaks indicates that there is a horizontal disparity of 14 pixels and no vertical disparity. (C) The image pair from Figure 6B, again reduced. (D) The cepstrum of (C). The peaks indicate no horizontal or vertical disparity.

Yeshurun and Schwartz [1987] describe a fast, one-pass method of determining the spatial disparity of a pair of images, based on the cepstral filter. The left and right images are juxtaposed horizontally to produce a new image of double width (see Figure 7A). The power spectrum of the log of the power spectrum (the "cepstrum") of the double image is then calculated. It can be shown that there will be peaks in the cepstrum at a location corresponding to the spatial disparity between the two images [Yeshurun and Schwartz 1987]. Finding these peaks is a simple matter.

The cepstral filter has several properties that make it particularly appropriate for the post-saccade vergence task. First, it is a one-pass operation that can be done relatively quickly. Two Fast Fourier Transforms are done, for a time complexity of $O(N\log N)$. Second, it can operate across the entire field of view and produce a single disparity vector. Third, it is very robust and can tolerate noise and large disparities.

After vergence with the cepstral filter, the views from the two cameras are subjectively well aligned (see Figure 6B). Of course parallax differences between the images force some objects to be out of alignment. The cepstral algorithm seems to do

a sort of voting operation, with each surface in view voting for its particular disparity. This behavior seems to contribute to robustness, since vergence on a relatively depth-uniform scene will not be prevented by the presence of a few objects at very different depths.

The algorithm is robust over repetition as well. A second application is not neccessary since it will generally report no disparity, with the exception of image pairs that started with a very large initial disparity. Disparities of more than half of the image size can often be measured with little or no error.

4.3.2. Future Work

We are currently investigating the use of the cepstral algorithm to images other than stereo pairs. One such application is an adaptive tracker. By calculating the disparity between small windows in two successive video frames from one camera, the retinal motion of an object can be determined. This retinal motion can be used to cause the camera to move to follow the object. In another application, the cepstral algorithm can be run on the time derivative of intensity. This will cause moving objects to become most significant in the disparity calculation.

A future project is to *use the cepstral filter as part of a distributed computation.* The fast cepstral filtering algorithm using the Euclid signal processing computer in MaxVideo can be integrated with the camera vergence system to provide not only the fastest vergence execution possible in our lab under the cepstral paradigm, but also the first example of a complex vision calculation being done in a peripheral processor. Running this calculation in parallel with significant computations in the MaxVideo processor (e.g., using FeatureMax to find (x,y) locations of features) or with significant computations in the Sun (e.g., shape analysis, robot control) would be our first example of significantly interacting, distributed vision processing.

Another project is to *produce a depth map, using the cepstral filter to establish image correspondences.* All our current binocular stereo applications trivialize the correspondence problem by domain restrictions (a blob is a set of pixels over a threshold, the blob centroid adequately defines the blob location, and there is only one blob to an image). More sophisticated features, such as those extracted from the image itself in the adaptive tracking paradigm, can be used to attack the correspondence problem. The cepstral filter provides a way to associate a disparity with an image patch without doing explicit object-recognition or segmentation. Thus its output can be interpreted as a set of "corresponding" pixels between two images. The corresponding sets are the cepstral patch in one image and the result of applying the disparity to the locations of all the pixels of the patch. The stereo geometry calculations could be applied to the pixels of the patch and to its "corresponding" disparity-shifted pixels to derive a 3-D location for the point set. If carried out for a set of patches that cover the image, the calculation yields a coarse depth map.

4.4. Efficient Implementation of the Cepstral Filter

4.4.1. Introduction

The two-dimensional cepstral filter provides an accurate and robust way to estimate the disparity between two images [Yeshurun and Schwartz 1987]. Unfortunately, it is computationally expensive. To compute a single disparity value, the procedure is:

1) Extract sample windows of size $h \times w$ from the left and right images. Splice them to form a single image of size $h \times 2w$.
2) Compute the 2-D Fourier transform of the resulting image.
3) Use the result to compute the power spectrum of the image, and take the log of each pixel.
4) Take the 2-D Fourier transform of the result.
5) Find the peak value in a subregion of the resulting image.

The peak values occur at $(\pm(w + d_h), \pm d_v)$, where d_h and d_v are the estimated horizontal and vertical disparities.

In one current application, cepstral filtering is used to maintain vergence. We derive a single disparity estimate for the left and right camera images and adjust the camera yaw angles to minimize the horizontal disparity. In theory the vertical disparity should be zero, but we expect small errors due to camera miscalibration, imprecision in the camera positioning system, et cetera.

Since we require only one disparity estimate per image pair, the cost of the cepstral filtering approach is not prohibitive. However, even single disparities are hard to compute at anything close to real-time rates. By analyzing the algorithm in detail, implementing carefully, and using special-purpose hardware, we have been able to compute disparities in less than a tenth of a second, which is more than adequate for our system. The remainder of this section describes our approach and experiences.

4.4.2. The Approach

In order to use disparities for vergence control, we must first decide what sample windows to look at in the camera outputs. If the sample windows are too large, they are likely to contain objects at different depths, which will lead to multiple peaks in the output image. If they are too small, two problems arise. First, they may not contain enough detail to provide an adequate basis for the disparity computation. Second, the two images may not overlap at all, in which case no disparity computation will be possible.

Having chosen sample windows, we must decide how densely to sample them. If the sampling rate is not high enough to satisfy the Nyquist criterion, the left and right images may contain spurious frequencies whose phase depends on the position of the sample window. If this occurs, the cepstral operator may produce meaningless

results. In general, denser sampling will improve the signal-to-noise ratio. The time complexity of the algorithm, however, is $O(n^2 \log n)$ for an $n \times n$ sample window, so the cost of denser sampling is relatively high.

We have so far made no attempt to optimize the sample window size and sample density parameters. The former we regard as a deep problem, which we hope to address seriously in a later study. Any solution will certainly involve an adaptive algorithm that adjusts the window size continously to suit changing conditions. In the experiments described below, the sample windows were usually the entire left and right camera images, though we sometimes varied the window size manually. Large sample windows worked reasonably well in our lab, which has little depth variation. As for sample density, one of us (Potter) determined empirically that reducing the 512×512 images to 32×32 by blurring and subsampling produced adequate results.

Our work to date, then, has consisted of optimizing the cepstral filtering operation on 32×32 subsampled versions of the original image. The remainder of this section describes our results.

4.4.3. The Hardware

The basic hardware configuration of the project is described in Section 3. Potter's empirical work used a very general cepstral filtering program running on the Sun 3/260 that serves as the main system controller. The implementation used floating point arithmetic throughout, and (despite the presence of a Sun/Weitek FPA) required 2.6 seconds to compute a single disparity. Optimization could have reduced this considerably, but did not appear likely to yield real-time performance. Also, we did not wish to burden the system controller with this type of computation-intensive task.

Instead, we implemented the second version of the cepstral filter on EUCLID, a single-board computer that is part of the DataCube MaxVideo board set. EUCLID consists of an ADSP-2100 digital signal processing microprocessor with 32K words of data memory and 16K words of program memory. The ADSP-2100 runs at 8 Mhz and has separate data and instruction busses; most instructions execute in one 125 nS cycle. Its instruction set and functional units are heavily optimized for digital signal processing applications. Half of the data memory is dual-ported to the VME bus, and supports zero wait state access by the ADSP-2100 concurrently with access by the Sun. The other half of the data memory is dual-ported to the DataCube Maxbus, so that regions of interest can be transferred in at video rates.

An important limitation of the ADSP-2100 is that the data memory address space is only 16K 16-bit words long. On the EUCLID board this space is divided into 16 1K-word pages. Fifteen of these pages can be mapped to 1K-word blocks in either half of local data memory, or to 1K-word or 1K-byte blocks of VME memory. The remaining page contains mapping and control registers and scratchpad memory. Because of the small address space, it is difficult to manipulate large data objects. This limited our sample density to somewhere between 32×32 and 64×64 samples per window.

The EUCLID board is programmed using a C cross-compiler, cross-assembler and linker running on the Sun host. These are not yet mature tools, and we had more than the usual quota of difficulty persuading them to work for us. We also found that the C compiler, while very useful for prototyping, did not make efficient use of the ADSP-2100's rather exotic architecture. It was necessary to convert all inner loops to assembly language subroutines. Fortunately the DataCube libraries provided a number of examples of well-written assembly codes, which we used as models.

4.4.4. A Preliminary Implementation

The derivation of the cepstral filter given in [Yeshurun and Schwartz 1987] is phrased in terms of continuous functions. As we said earlier, it is based on Fourier transforms of an $h \times 2w$ image formed by splicing two windows together. Unfortunately we cannot simply apply a 2-D FFT to this window. A discrete Fourier transform of this size can only handle frequencies in the range $-h/2$ to $h/2$ vertically and $-w$ to w horizontally. Positive horizontal disparities will produce peaks falling outside this range. (Actually, since discrete FFTs are circular, positive disparities will wrap around to the opposite ends of the horizontal frequency axis. The result will be that we can determine the magnitude of the disparity but not its sign.) The obvious solution to the problem is to widen the image by padding with zeros. For the cepstral of two 32×32 sample windows, for example, we padded the image to 32×128 by appending 32×32 arrays of zeros to each end.

After constructing the input array the cepstral algorithm requires us to perform a 2-D FFT, take the log of the power spectrum of the result, and perform a second 2-D FFT. A multidimensional FFT is performed by applying 1-D FFTs along each dimension of the input array. In-place FFT algorithms require either the inputs or the outputs of the transform to appear in scrambled (bit-reversed) order. We avoided the overhead of scrambling inputs or unscrambling outputs by adopting a strategy used in linear filtering. We first transformed each column of the input using a decimation-in-time (DIT) FFT that expects normally ordered input and produces scrambled output. Since each column was scrambled identically, this left the data correctly ordered within each row; the rows simply appeared in a different order. We then transformed each row by the same DIT FFT. This completed the first FFT, but left the data scrambled in a very complex way. However, taking the log of the power spectrum is a point operation, so the fact that the points were out of order was unimportant. For the second FFT we used a decimation-in-frequency algorithm that expects scrambled input and produces normally ordered output. Applying this transform first to the rows and then to the columns undid the scrambling produced by the first FFT, leaving us with a correctly ordered output image.

Computing the power spectrum from the Fourier transform is trivial, but taking the logarithm at first seemed likely to present problems on an integer machine. The standard derivation of the cepstral filter uses the natural log (base e), but we noticed that the base is immaterial to the algorithm; all that matters is that $\log ab = \log a + \log b$, which is true for any base. We found that we could estimate $\log_2 n$ (in fixed point, for any integer $n > 2$) to within 2% by counting the bits in n and then linearly interpolating between the two nearest powers of two. We later found that the algorithm works quite well even if the interpolation step is omitted, so our final

implementation simply counts bits. The ADSP-2100 has a NORMALIZE instruction, so counting the bits requires only two or three instructions per point.

As a final optimization, we noted that half of the columns in the input image contain only zeros, so there is no need to compute their FFTs. Also, the final set of column transforms only needs to be done in the region that will be searched for peaks. This region consists of the columns representing frequencies between $w/2$ and $3w/2$. Thus we can eliminate 5/8 of the column transforms.

We found that the C code required to find peaks in the output consumed only a small fraction of the run time, so we did not optimize it. Using the optimizations described above, EUCLID computes the cepstral disparity estimate for 32×32 windows in 120 milliseconds, not counting the time required to load the image into EUCLID data memory. Figure 8 shows typical inputs and outputs of the implementation described above. For display purposes, all of the columns in the output have been computed, and the result has been shifted to put frequency (0,0) in the center. We have also taken the log of pixel values in order to reduce the relative brightness of the central peak. The two bright dots to the upper right and lower left of the central peak are the disparity peaks. The apparent noisiness of the cepstral output is an artifact of the reproduction process; in fact the disparity peaks stand far above the noise background, and can be extracted with very high reliability.

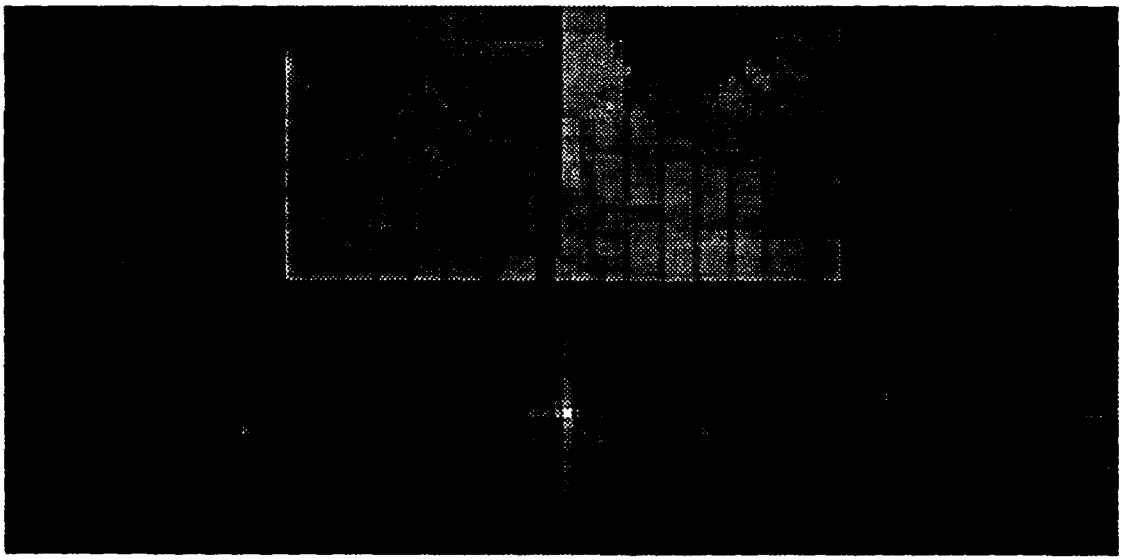


Figure 8: Inputs (above) and outputs (below) for the preliminary implementation. Note the large patches of zeros at either end of the input.

4.4.5. The Final Implementation

Although our first EUCLID implementation gave acceptable performance, we found it aesthetically unsatisfying in one respect. Given that time and memory were scarce resources, the need to pad the image with zeros was very annoying. The extra work

did nothing to improve the signal-to-noise ratio; its sole purpose was to provide enough frequency range to handle the expected output.

After some thought we realized that the range problem can be gotten around in another way, at least for our application. Recall our previous discussion of what happens if we apply a 2D FFT to an unpadded cepstral input image. If the horizontal disparity is positive, the output peak that should have appeared at $(w+d_h, d_v)$ will wrap around to the other end of the spectrum, appearing instead at $(-(w-d_h), d_v)$. At the same time, the peak that should have appeared at $(-(w+d_h), -d_v)$ will appear at $(w-d_h, -d_v)$. The transform output will then be indistinguishable from that produced by disparities of $(-d_h, -d_v)$. But now suppose that (as in our case, or in the case of stereopsis) we can assume that the vertical disparity is small--specifically, that it is in the open interval $\pm h/4$. We can "tag"" the disparity peaks by introducing an artificial vertical disparity of $+h/4$. That is, we tranform the right-hand sample window by moving row m to row $(m+h/4) \bmod h$. Discrete Fourier transforms wrap around at their borders, so we will not lose any samples and the signal-to-noise ratio will not suffer.

After the cepstral transform, we locate peaks in the usual way. We know, however, that the vertical disparity should be positive. If we find a negative vertical disparity, we know that the horizontal disparity wrapped around, and that the correct disparities can be obtained by flipping the signs of the measured disparities.

This algorithm runs on EUCLID in 60 milliseconds for our standard 32×32 sample windows. (In our vergence application EUCLID spends an additional 20 milliseconds fetching the sample windows from the ROIStore frame buffer.) Figure 9 shows sample inputs and outputs. A final optimization (which we have not done) would be to take advantage of the fact that the FFTs in the cepstral filter operate on real data. We would use a real-data transform on the image rows, and then (since the output is known to be symmetrical about the origin) transform only half of the columns. We expect that this would cut the run time approximately in half, to about 40 milliseconds.

4.4.6. Conclusion

We have shown that by implementing carefully and eliminating redundancy in the algorithm we can compute single cepstral disparities at close to the RS-170 video frame rate. In addition, our work with the cepstral filter has helped us to develop some intuition about why it works so well. Yeshurun and Schwartz [1987] view the transform as extracting a periodic term in the power spectrum of the sum of the original and shifted images. We prefer to think of it as a generalization of autocorrelation. Our argument is as follows:

Ignore for the moment the fact that the cepstral input consists of the left and right images spliced together. What does the rest of the algorithm do? It is well known that the power spectrum is the Fourier transform of the autocorrelation function. If we skip taking the log of each pixel, and replace the second Fourier transform with the inverse transform, then we will compute exactly the autocorrelation. But the Fourier transform of any function is just the complex conjugate of the *inverse* Fourier transform of that function. Since the autocorrelation

Figure 9: Inputs (above) and outputs (below) for the final implementation. Note the circular upward shift applied to the right-hand image, and the lack of zeros in the input.

function is even-symmetric, its Fourier transform is strictly real--it is its own complex conjugate. Therefore the second Fourier transform might as well be an inverse Fourier transform. If we omit the log step, then, the filter just computes the autocorrelation function.

What is the effect of taking the logarithm before the inverse Fourier transform? For any *particular* input image, it is equivalent to multiplying each point in the transformed autocorrelation function by a constant; that constant simply happens to depend on the image. This implies that the cepstral filter is equivalent to a linear filter applied to the autocorrelation function, with the restriction that the filter is tuned to the particular image. At present we do not have a clear understanding of what this filter does. We hope to begin a serious study of this question in the near future. We will look at the problem empirically as well as analytically--for example, we might compute equivalent linear filters for several images and see how they differ. Our current intuitive answer is the following: the log function makes small numbers a little smaller, but it makes large ones a lot smaller. Thus for two signals of equal energy, it will favor the one whose spectrum is broader (and hence has smaller peak values.) The signals with the broadest spectra, of course, are impulses; so the logarithm tends to enhance spikes in the autocorrelation function. The cepstral filter can be thought of as modifying the image to make its autocorrelation function cleaner and less ambiguous. We look forward to testing this hypothesis in the near future.

5. Animate Vision

This section describes more complex visual capabilities that involve the active participation of the seeing agent: depth from actively-initiated parallax, segmentation using actively-initiated head motions, and integrated visual and motor behavior.

5.1. Real-Time Relative Kinetic Depth

Kinetic depth is the sensation of visual depth that one gets when moving the head while fixating a target [Ballard and Ozcandarli 1988]. Objects in front of the fixation point appear to move in the opposite direction to the motion while objects behind the fixation point move in the same direction. The apparent velocity is proportional to the distance from the fixation point (Figure 10).

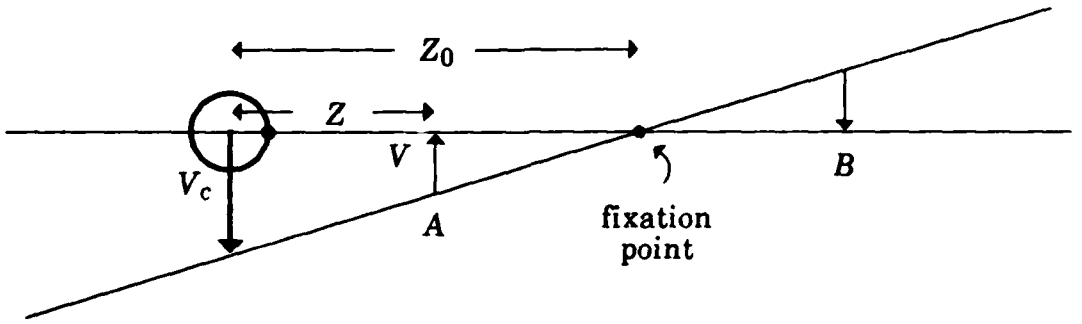


Figure 10: Kinetic depth obtained by making head motions with respect to a fixation point (A) appear to move in the opposite direction to the commanded motion V_c, whereas points behind (B) appear to move in the same direction.

The experimental setup is shown in Figure 11. The computation consists of: (1) a tracking mechanism that holds the camera fixed on an environmental point, and (2) a depth computation mechanism. The tracking mechanism currently finds the centroid of a light object against a dark background and uses its movement in image space to compute an error signal for the eye motors. The depth computation is based on the formula giving relative depth as a function of image spatial and temporal derivatives, I_x and I_t,

$$Z = AI_x / (I_x + BI_t),$$

where A and B are constants. To create I_x the image is convolved with an edge template (thus only horizontal motion is sensed in this implementation). To create I_t two successive video frames are subtracted. The values of I_x and I_t are used as input to a lookup table that computes Z using the equation above.

Figure 12 shows a video frame captured from the moving input image and the corresponding depth output. To make the result visible, the quantitative depths in

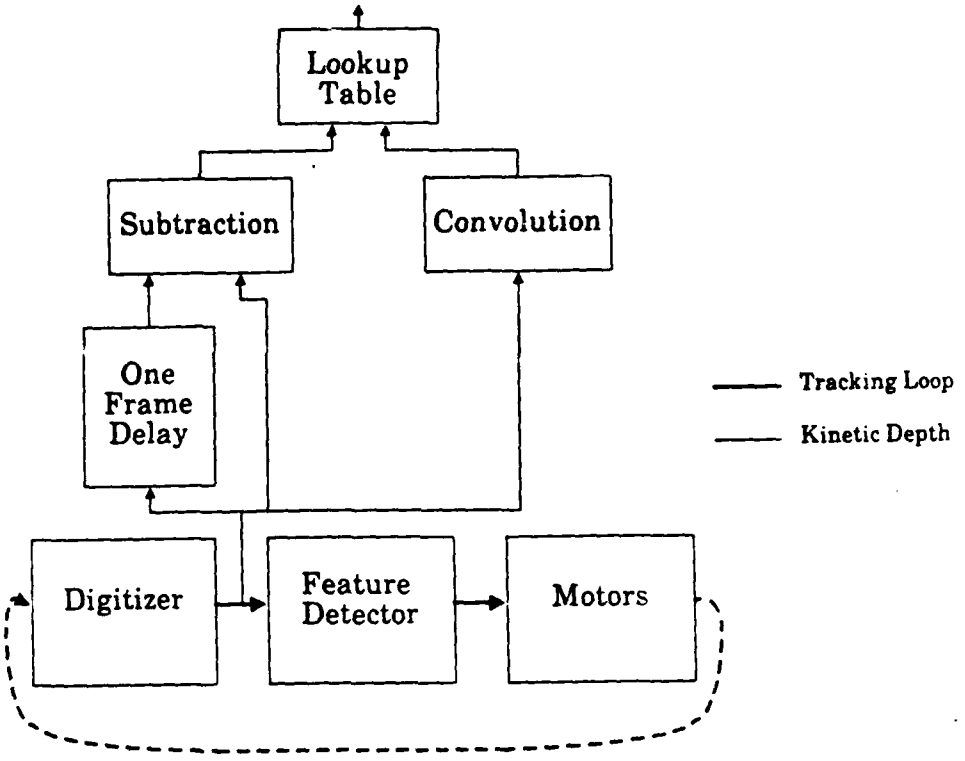


Figure 11: Block diagram for real-time computation of kinetic depth. Blocks correspond to boards in the MaxVideo system and the motor controllers.

the output image are binarized: those behind the fixation point are coded as black, those in front as white.

5.2. Dynamic Segmentation

The ability to maintain fixation while the head moves allows the visual isolation of objects from the background (or partially obscuring foreground), and also considerably simplifies the image to be analyzed. Figure 13 illustrates the phenomenon. The processing is simply to compute a weighted average of input images, captured continuously as the robot moves. At each frame, the moving weighted average is MovAve(t+1) = k(MovAve(t)) + (1-k)Image(t). For the examples in Figure 13, k = 0.9. This capability is being implemented to run at frame rates on the MaxVideo system.

The interaction of this capability with other visual capabilities is interesting. The same head motion can, as we have seen, create a full-frame relative depth map from the kinetic depth effect, and also isolate the fixated object from the background to make its extraction by vision operations easier. When the cognitive system is paying attention to the fixated object, say to do recognition, the background recedes from active consideration and may not necessarily be kept updated with the relative 3-D

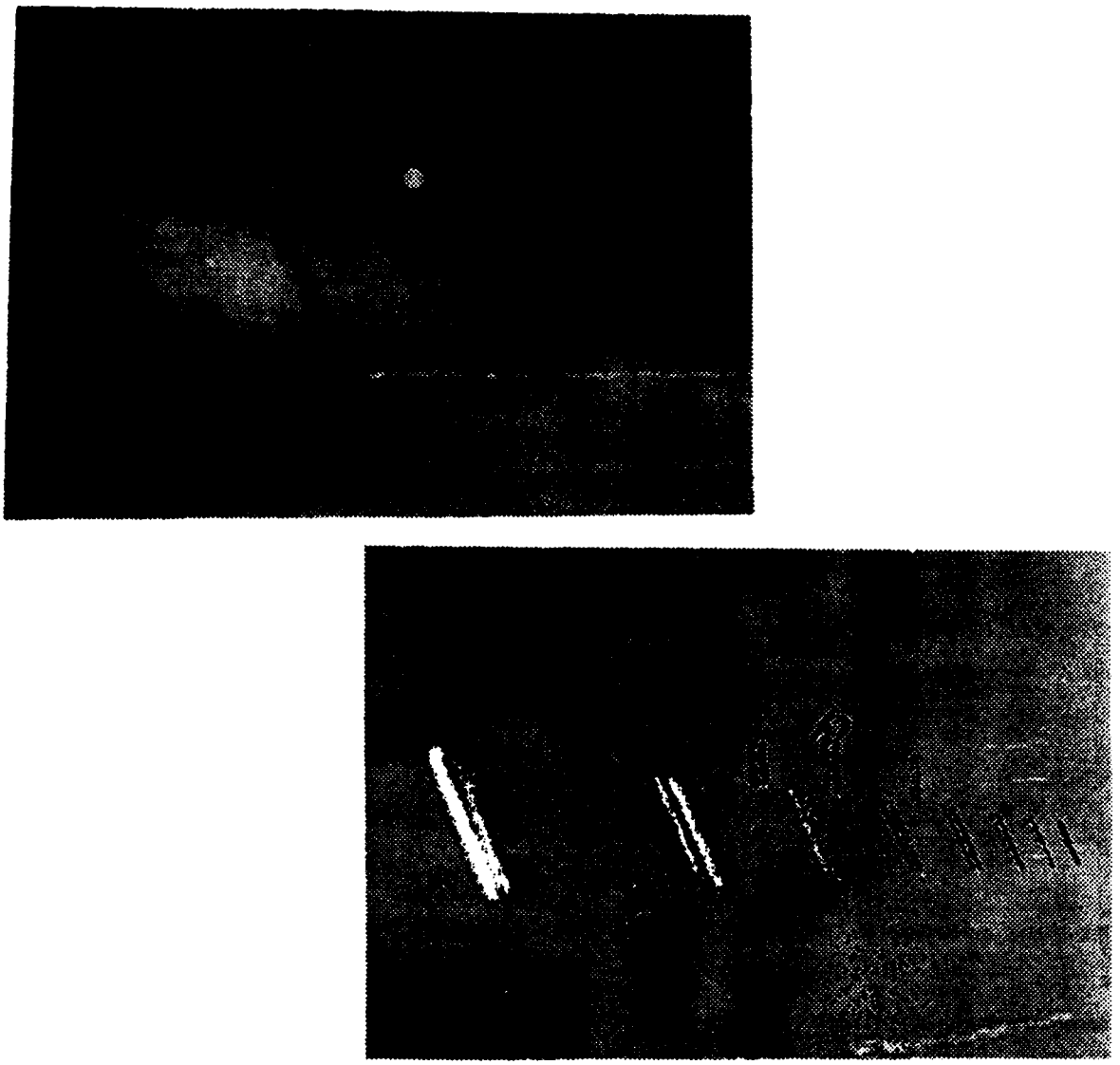


Figure 12: (A) Original grey-scale image of a hand-made ruler on a table with a piece of chalk. The chalk is fixated. The ruler's markings are approximately 10 cm apart. (B) Kinetic depth image taken from output of lookup table and thresholded to increase visibility of effect.

positions of its features. This indicates that dynamic segmentation is receiving more computational resources. However, when attention is directed to the whole scene, a static world seems to be a set of rigidly-connected, three-dimensional objects. This indicates that the 3-D aspect of the world is being maintained and the processing of depth information (among other things) is getting more computational resources.

Active segmentation can be driven by images stabilized by the adaptive tracking algorithm (Section 4.2), or by compensatory reflexes. We are proposing to develop

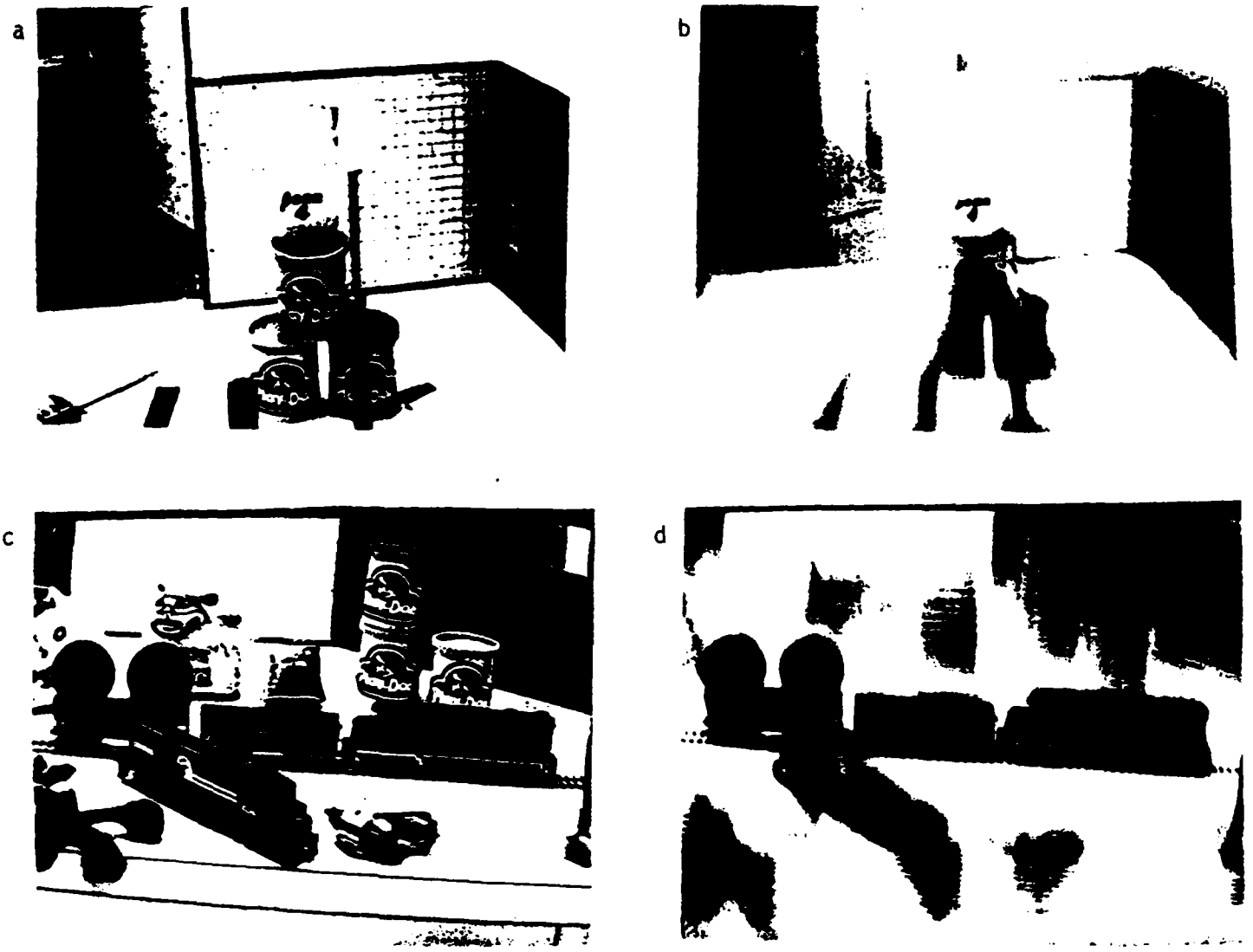


Figure 13: (A) A scene honoring the University of Pennsylvania, a pioneer in active vision. (B) Fixation while moving back from scene of (A). (C) A train scene. (D) Fixation on train in scene (C) while accommodating motion in altitude. (E) A doll scene. (F) Edge operator applied to (E). (G) Early stages of computing moving average in scene (E), accomodating motion in azimuth. (H) Later stage in moving average. (I) Edge operator applied to (H). (Figure continued on next page.)

this idea toward practical use in object recognition in scenes involving robot or object motion. Of course the object recognition (indeed the segmentation) problem will not be solved. However, not only is it necessary to attack the basic problems inherent in recognition during object and observer motion, but also we should like to

e
f
g
h
i

demonstrate that motion, when used correctly is a basic part of the solution rather than just another part of the problem.

There is no complete theory of how we (and presumably animals) perceive a stable world despite eye movements and body movements that keep the retinal image of the world constantly changing. One reasonable and partly-correct theory is that since eye and body movements are cognitively commanded, the visual system has enough information to anticipate and compensate. However, the system is more robust than that and can compensate for some unanticipated retinal shifts as well ([Hein and Jeannerod 1983] presents several theories).

The stable world is an important representation [Feldman 1985], especially for actively intelligent entities that must interact with it. To construct a representation that is stable across eye movements is not necessarily easy. In the case that input data is not put immediately into a world (LAB) coordinate system by low-level routines like stereo, then matching the structures seen in separate views is needed to maintain a stable (rigid, coherent) representation of the world. *Peripheral vision* is a promising mechanism, even for robots, to integrate information over movements. It is easy to demonstrate [e.g., Shafer 1988] that an impossible quantity of information is needed to maintain high-resolution, "all- foveal" approach to a wide (60°) visual field without some form of automation: 10^{14} bits per black and white image, or a rate of a million megabytes a second (conservatively). Wide visual fields make continuity between eye movements easier to maintain, since the amount of overlap is considerable between the full visual field (but is usually nonexistent between different foveations). Last, multi-grid techniques and multi-resolution approaches to visual processing are of highly proven utility [e.g., Terzopoulos 1983, Hanson and Riseman 1988, Brown et al. 1982].

Thus we propose to *use peripheral vision to help integrate information over eye movements*. The idea is to perform low-resolution visual calculations in the periphery (provided by a wide-angle camera mounted close to the "foveal" camera) in parallel with foveal processing. Peripheral vision is useful for directing attention and for producing preliminary low-resolution representations of image structures. When a point from the periphery is foveally fixated, high-resolution processing already has a context and a start on its work. This amounts to application of a multi-grid approach on selected subsets of the visual field, building up areas of better information about the world with each fixation. This is an idea we began to investigate in the context of Markov Random Field models of low-level visual operations [e.g., Chou and Brown 1988], and which still seems promising even using much simpler methods of information integration.

5.3. Future Work: Vergence, the VOR, and Stereo

The current vergence operation is based on a feedback signal from images (the cepstral filter-generated disparity of image patches). It assumes nothing about (in fact it derives) the correspondence of individual pixels. If vergence is used to derive eye orientations, and if it guarantees that certain patches of images correspond, then a binocular stereo algorithm can be run to determine the three-space location of the pixels in the patch. Then, head motions can be anticipated and fed forward through the known relation between scene points, robot configuration, and camera angles, to

yield eye motions to compensate for the head motion (implementing a functional equivalent of the VOR). If the compensations are not perfect, the vergence operation will notice a remaining disparity and can supply the necessary correction. The cooperation between the two systems can help both: incorporating the vergence capability with the stereo capability assures that the corresponding points in the two images stay near the center of the images, thus obtaining optimum calibration accuracy. With large objects, vergence provides a mechanism to keep the object in view despite motions. The VOR can help the cepstral vergence mechanism by keeping disparities within small bounds.

5.4. An Integrated Exploration, Manipulation, and Inspection Workbench

The Ball Mapping Workbench (BMW) is a system that runs in the Active Vision Lab. It deals with the domain of static objects and uses capabilities for deriving and using the estimated 3-D locations of objects in space [Brown and Rimey 1988]. BMW operates in near real time (the principal slowness is in the robot control). The speed is possible largely because of the speed of the visual processing in the MaxVideo system.

BMW allows the robot to explore its environment for objects, locate them in space, touch them, and simulate a VOR-like mechanism using robot movements. BMW uses a modular design that encourages re-usability and expandability of software. It can serve as a platform for many further integration and experimental efforts, such as those in the following section.

5.4.1. BMW Organization

The overall design and various components of the BMW are discussed in this section. BMW uses as components several previous projects, including several MaxVideo routines and robot control software ported from Purdue, as well as new projects completed this summer by Chris Brown, Edmond Lee and Ray Rimey. Besides new robot vision and motion capabilities, a substantial contribution was the framework to organize and control the components. This description concentrates on the components and organization not discussed elsewhere in the report. Two important characteristics of BMW organization are the following.

1) *Centralized Control.* All aspects of the BMW are directly controlled by a Sun workstation. This includes control of the Puma robot, eye motors, and MaxVideo image processing boards. The "explore and touch" mode of the BMW runs completely without human intervention once it is set up and started.
2) *Modular Design.* BMW is designed in an intuitive modular fashion. In the manner of abstract data types, each major subsystem has a single data structure that stores the current state of that subsystem. These subsystems also have a set of functions that operate on the subsystem and maintain the stored state. The data structures and modules are similar to those found in [Brown and Rimey 1988].

5.4.2. BMW Components

The current major subsystems are:

1) Puma
2) Head (includes eye motors and cameras)
3) Image model in the form of 2-D blob lists
4) World model in the form of 3-D blob list

There are also a number of miscellaneous software modules in the system.

1) Control of the MaxVideo image processing boards
2) Extracting multiple blobs from an image
3) Estimating object location constrained by fixed depth
4) Two versions of object location via binocular stereo

5.4.3. Current Problem Domains and Modes of Operation

The "interesting" objects for the current BMW are racquet balls (blue) before a white background (the walls of the lab). Balls are located monocularly using binary image analysis. The location of a ball in 3-D space is estimated using binocular stereo and the two cameras mounted on the head. BMW currently implements an "explore and touch" mode. The robot will "explore" a volume in the Lab and locate all the balls in that volume. The current exploration module performs a sequence of horizontal translations of the robot head. After exploring, BMW touches each ball with a poker mounted on the head to prove that it has located the balls correctly. BMW also contains an "inspection" mode that causes the robot to view an object from a variety of different head locations while keeping the object foveated in one of the cameras. This capability is based on the TOOL coordinate system definition facility in the VAL system, uses only the six robot joints and does not incorporate camera movements.

Extracting Multiple Blobs from an Image. The task of finding multiple blobs in an image consists of two separate subtasks: (1) obtaining a list of image points belonging to blobs, and (2) grouping these points into several blobs. The first step is performed in frame time using the MaxVideo board named FeatureMax, which emits in scan-line order the (x,y) image locations of pixels that have certain characteristics (in the current BMW, they are above a threshold). The second step is performed in software on the Sun using a scanline blob extraction algorithm. The basic idea of the algorithm is simple. The image points are run-length encoded. Then the image is processed one row at a time, starting at the first row. When processing any given row the program has a list of blobs it has created from all previous rows. When processing a row the runs in the row are compared for overlap with runs in the previous row. Since the overlapping runs in the previous row already belong to blobs it is easy to merge the runs in the current row into the appropriate blobs. It may also be necessary to create new blobs or merge existing blobs. The algorithm computes the centroid of the final blob. This type of algorithm is inherently capable of computing many useful blob descriptors (moments, compactness measures, Euler number, etc.) on the fly. A comprehensive version of it

has been implemented in VLSI, and is available as a product that emits exhaustive descriptions of multiple blobs in real time.

"Explore and Touch"--Flow of Control. BMW executes its "explore and touch" function as follows (more complex exploration strategies, and different interleaving of exploring, touching, and inspection are of course easy to implement).

```
Initialize all major subsystems
For all locations in the exploration sequence (currently a set of horizontal translations)
    Move robot there
    Left camera: acquire image, extract multiple blobs, store in 2-D blob list
    Right camera: repeat above image analysis sequence
    Perform stereo computation
    Try to add 3-D blob to world model
End for all locations loop
Move robot to touch each 3-D blob in world model
Optionally inspect each 3-D blob from a continuous trajectory of locations.
```

Object Location Constrained by Fixed Depth. Early on in the BMW project we needed the ability to estimate the location of an object, but we were not ready to implement stereo. Hence, we constrained objects to lie in a plane perpendicular to the camera axis and a known distance away. In this situation it is easy to estimate the object's remaining two degrees of freedom.

A simple calibration procedure is used to create a lookup table of image (i,j) to world (x,y) pairs. The calibration tool is a flat black board containing a grid of evenly spaced spots. The tool is positioned perpendicular to the camera axis at the specified distance. The (x,y) offset of the tool with respect to the camera axis must be measured. An image of the tool is taken and the image (i,j) location of each dot is determined. These (i,j) locations and the corresponding world (x,y) location of each dot (already known) forms the lookup table.

The lookup table is used as follows. An image of an object is taken and the (i,j) location of that object in the image is determined. Locate the three (i,j) points in the lookup table that are nearest the object's (i,j) value. Interpolate between the three points in the lookup table to determine the (x,y) value for the object. The z value of the object is simply the fixed depth at which the calibration was originally performed. Thus the (x,y,z) location of the object relative to the head (actually, in FLANGE coordinates [Brown and Rimey 1988]) is determined.

The Stereo Algorithms. We have implemented two stereo algorithms [Duda and Hart 1973, Brown and Rimey 1988]. The algorithms can produce the location of an object in laboratory coordinates, for the world model, or in FLANGE coordinates, which are suitable for immediate use in manipulation. Objects are located in each image using the binary blob extraction software described above. The current implementation requires only one object in the fields of view. An incorrect stereo correspondence is detected via a threshold on the location estimate residual. Another test requires location estimates to be in front of the cameras.

We performed a crude sensitivity analysis to variations in blob location and camera angle. The procedure was to compute the location estimate using the original values for blob location and camera angles, and then to repeat the calculation with the blob location perturbed by one pixel, and then again with a camera angle perturbed by one degree in azimuth. Assume an object is one meter distant. One pixel deviation caused location devation by (0.58,1.5,14) mm. And one degree azimuth angle deviation caused location devation by (19,0.52,5.9) mm.

World Model. Locations of objects are generally estimated relative to the FLANGE coordinate system that is mounted on the head. Since the location of the robot head can be obtained from the robot controller it is a straightforward task for BMW to convert object locations from FLANGE to LAB coordinates. Finally BMW tries to add an object to its world model (list of 3-D blobs). A new object is added only if another object does not already occupy that volume in the world model.

Touching Objects. The robot touches an object by causing its "tool" to move to a specified location in space. This movement causes a collision between the robot tool and object. The robot controller allows the "tool" to be defined arbitrarily. So, the tool is defined as the far end of the stick attached to the head. The location of the object to touch is simply taken from the world model.

Inspection Module. BMW contains an inspection module that has an eye-stabilizing capability which allows the robot to move the head but to have the robot "wrist" compensate to keep one of the cameras (or both, if they are verged) fixated on a point in space. Thus the system can view an object from a variety of different head locations while keeping the object foveated in one of the cameras.

5.4.4. Conclusions and Future Work

BMW is the first integrated system in the Robot Laboratory to use Sun-based Puma control, the first to locate multiple objects in one image in real time, and the first to sense and interact with the world in quantified three-dimensional coordinates. It successfully incorporates several previously-written modules. BMW has proved a useful and flexible experimental platform for investigations into active vision (currently, interacting with static objects).

Originally BMW was conceived as a modification of the Spot system [Coombs 1988]. Functionally the tasks performed by the two systems are closely related. However, Spot was itself built on Rover [Coombs and Marsh 1987]. Rover was designed to run on first-generation Datacube software, which has no real-time image processing capabilities. Rover further was a showcase for a sophisticated real-time priority scheduling system for allocating computational resources, and many of its basic routines were implemented as schedulable processes. BMW has a much simpler control and resource allocation problem. Thus while the image-processing aspects of Spot and Rover have been superceded by hardware advances, the resource allocation issues are a lurking problem for which Rover has provided an answer to which we shall probably turn before long.

The experience of creating and using BMW was generally painless, but following are a few suggestions to smooth over the few bumps in Integration Road and to make driving even easier.

The largest problem has been with the MaxVideo software. Currently the state of the art is to borrow others' software. Because of the lack of standardized interfaces to the boards, or even standardized initialization sequences, such borrowed code fragments are incompatible or difficult to reconcile with one's current goals. Working code is currently the only reasonable way to communicate MaxVideo lore (one example is the excellent software by Dave Tilley that performs tracking using a Digimax, ROIStore, and FeatureMax boards). However, problems arise when one must integrate or even understand pieces of projects from several individuals. The interface mentioned in Appendix 1 would be a big step forward. Also, it would help greatly to have the following.

1) A standard for the most basic cable configuration. Fancier cable configurations could then be built on top of the basic one.
2) A set of standard functions which initialize and properly set up the boards. People who want to do something slightly different would call the basic init routine and then, in their own code, change the few things that they want different.
3) A set of standard ROI parameters.

Although BMW was designed and built to be a workbench for future integration, its data structures and interfaces could use some further improvement. BMW has already taught us a fair amount about user abstractions of our basic robotic capabilities, and as we learn more our software should likewise evolve in sophistication and elegance.

Now that BMW exists there are a few major directions which could be taken for further work. These directions are of interest in themselves, and all would further the potential of the laboratory for sophisticated use in organized projects.

1) Our capability for real-time object location must move beyond binary image blobs to real objects in a real scene--nothing artificial. The adaptive tracking project should be integrated into BMW as soon as possible.
2) We shall soon need the ability to control two or more systems at the same time. For example, move the robot head while tracking an object with eye movements, while computing 3-D locations via stereo. In short, we need to explore concurrent processing in a more serious way. Brown has suggested an evolutionary approach in which modules communicate through shared memory: their flow of control can then be flexibly modified through synchronization primitives. Concurrency is also available through multi-processing (on the Euclid or Butterfly computers, for example).

6. High-Level Control

The cognitive aspect of robot control is being approached by two complementary pieces of work. The cognitive levels will command the lower levels, and can influence or even override the "reflexive" behavior at the lower levels.

6.1. Planning and the Real World: Parallelism, Sensing, and Uncertainty

We have developed a planning domain based on model trains, ARMTRAK, which captures the conceptual simplicity of the blocks world while extending it sufficiently to exercise planners with more realistic capabilities. The inputs to the domain are simple: switch settings and engine power; but many different scenarios can be generated from these simple inputs. Because the trains movement is independent of the planner, plans which succeed at one point at may fail as the train progresses. The planner must therefore take into account temporal constraints on the generation and execution of its plans. Moreover, model trains are notorious for falling off the track. Therefore, plans may succeed or fail due to unexpected effects of the primitives. Finally, in certain cases there may be no plans which achieve the desired goal, but which may partially achieve that goal. In this case, all possible plans fail, and the planner must select the most useful plan from the set of failing plans.

Traditional planning has concentrated on issues of correctness and performance in controlled environments. Planning of this type has been well served by the blocks world. This planning domain does not, however, address several new problems raised by attempts to build robots which plan actions in the real world. While planning domains must be sufficiently simple for humans to reason about, they must also be sufficiently complex to exercise planning programs. Though the blocks world admirably captures the first requirement, it fail on the second. The blocks world makes five simplifying assumptions that effect the nature of the planners which operate on it. These assumptions are:

1) only the planner has effects on the world
2) the planner's actions always cause expected effects
3) only one action can occur at one time
4) a goal must be completely achieved for a plan to succeed
5) complete information is always available

Due to these simplifying assumptions, the blocks world fails to support sufficient complexity to exercise planning under three major categories of constraints:

1) real-time planning
2) planning with the possibility of unexpected results from of primitive actions
3) selecting between equally attractive plans

ARMTRAK consists of model train layout coupled with a sensorium. A set of commands manipulates the layout; the sensorium satisfies a set of queries. The layout consists of track and trains. The track comes curved or straight sections. The curved sections are categorized by the radius of the circle which the track describes.

Switches are sections of track which are both curved and straight. The state of the switch determines which of the two alternatives is taken. Decouplers are sections of track which can uncouple cars. Track is a sequence of track sections. Trains are composed of a locomotive and cars. Signals sent to the locomotive specify the power supplied to the engine. The velocity of the train is determined by its power, its weight, and the grade of the track it is on. When a car backs into another car the cars connect. Commands can be sent to the switches, the trains and the decouplers. Commands to the switches change the state of the switch; commands to the engines change the engine power; commands to the decouplers uncouple the cars.

ARMTRAK will have advantages over the blocks world due to increased flexibility along four axes. First, ARMTRAK will follow the actual passage of time more closely. Flexibility along this axis will allow exercises involving planners which maintain complex temporal models. Next, complex goals can be specified allowing the use of planners which attempt to achieve optimal results rather than blindly achieving a goal set for it. Also, the domain will have a sufficient number of elements so that realistic conversations can be generated about the domain allowing the use of discourse system involving plan recognition. Finally, unexpected results can occur allowing the use of planners designed to take advantage of serendipity.

Deviser [Vere 1983] is a planner which allows plans with temporal parameters. This is useful in situation in which the planner must interact with scheduled events. Such events are not possible in the blocks word, but they are possible in ARMTRAK. Allen and Koomen [1983] describe a planning system which allows even more complete temporal plans. In particular they mention the possibility of planning overlapping events. Such events are possible only in situation in which multiple action can occur simultaneously. Multiple trains in ARMTRAK will allow such simultaneity.

Feldman and Sproull [1977] and Feldman and Yakimovsky [1974] discuss choosing between different possible plans by using decision theory. Such choices are possible only if there is a way in which utility can be associated with goals, and costs associated with actions required to achieve these goals. Utilities and costs cannot be associated with the goals and actions of the blocks world in a natural way. By extending intuitions about real trains to the model trains world, such an associated can be achieved naturally. Costs are simply the length of the track traversed; utilities are associated with the cargo held in the cars.

Plan recognition [Allen 1987, Kautz 1987] is frequently used in natural language understanding systems to resolve semantic ambiguities. The need for plan recognition arises only under circumstances which are sufficiently complex to require disambiguation. The blocks world is not sufficiently complex. ARMTRAK does provide sufficient complexity. For example, suppose the goal given to the planner was to move train 1 to location A by time t. Upon realizing that this goal cannot be achieved, the planner might suggest an alternative plan, say, moving train 2 to location A, "because it also has cattle cars." The planner has recognized an implicit plan in the goal statement and has generated a possible plan to achieve the implicit plan.

Finally, reactive planners [Firby 1987, Agre and Chapman 1987, Georgeff and Lansky 1987, Brooks 1987] make use of unexpected events in the course of executing their plans. These results may be deleterious such as the derailment of a train necessary to the execution a plan, or serendipitous as in the derailment of a train which blocks a train necessary to the execution of a plan.

Two versions of ARMTRAK are being implemented: a simulation, ARMTRAK-s, and a set of trains coupled to the sensorium provided by the active vision lab, ARMTRAK-r. The simulation allows rapid prototyping of planners, and experimentation with problems posed by different layouts. Because building model train layouts requires an investment of effort, the availability of the simulation allows users of the systems to perform quick experiments to answer specific questions. On the other hand, simulations invariably involve simplifying assumptions. The availability of the real trains and the sensorium provided by the vision lab insures that the results obtained through the use of the simulation are accurate. The simulation will allow easy experimentation; the real trains will insure that these experiments are honest.

The ARMTRAK-s displays the following characteristics:

1) The information describing the simulation will be maintained as a relational database allowing easy modification to the queries.
2) Planners will link to ARMTRAK-s through a series of library functions which return values on query, or change the state of the simulation on command.
3) The communication between ARMTRAK-s and the planners will be maintained by stream based sockets allowing the use of planners on other machines. Separation of the planners from the simulator will also insure that planners built using it will work with the real model trains.
4) The overriding concern in the development of the simulation is that it will accurately model the real model trains, and that planners developed using the simulation can easily be applied to the real trains. Another concern is to maintain separation between the planners and the simulation.

The ARMTRAK-r is designed to appear as much like ARMTRAK-s from the planner's point of view. This similarity is insured by the following characteristics.

1) Planners will link to ARMTRAK through library functions with the same names as the function in the simulation.
2) ARMTRAK will be implemented on a Sun workstation communication with planners through stream based sockets.
3) The overriding concern in the development of the real model trains is that the command and query interface is the same as that provided by the simulation.

The benefits to vision research will flow in two directions from ARMTRAK. First, ARMTRAK's primitives will remain stable providing a high level goal for the integration of vision routines. Next, because ARMTRAK will be used primarily as a testbed for planning, these vision routines will be thoroughly exercised by users

other than their designers. This use will provide valuable feedback both on the robustness and the usefulness of these routines.

6.2. Multi-Tasking and Planning with Parallel, Tightly-Coupled Modules

6.2.1. Data Flow, Activity Units, and Planning

One of our research projects is aimed at exploring computational models for organizing the control systems of autonomous robots. We have developed a control model that is based on the tightly coupled interaction of simple computing units. The model was designed to address four major concepts.

1) The model provides mechanisms for organizing and building in behavioral knowledge.
2) The model supports truly concurrent activity and provides mechanisms to manage physical resources (e.g., sensors and effectors).
3) The model is based on a continuous control system metaphor, allowing behavior to be tightly coupled to the environment [Kuo 1982, Franklin and Powell 1981],
4) The model supports layered control [Brooks 1987].

Although we realize that there is far more to intelligent activity control than these four concepts, (For example the model has no learning mechanism, no anticipation mechanism, no long-term "symbolic" planning, and only a simple form of memory), we feel the model embodies the essential components needed to build robust robot control systems capable of exhibiting simple forms of intelligence (analogous to low order vertebrates). Phylogenetic and anatomic evidence suggest that nervous systems exhibiting complex cognition are elaborations on simpler systems. We suggest that autonomous robot evolution is destined to follow the same course.

A control system based on our model conceptually consists of a network of cooperating units, called AUs (activity units). Logically, each AU is a processor. Thus at any instant many AUs can be running. An AU that is running is said to be active. There are two types of AUs: tonic AUs and phasic AUs. Tonic AUs are always active, phasic AUs only run when signaled by other AUs. Each AU is responsible for performing a specific task. What that task is and how it is performed is determined when the AU is defined. Tonic AUs may be used to implement sensory and monitoring processes, since they are constantly active and can continuously process incoming sensory data. The effect is that such an AU is constantly receiving an influx of sensory data and generating an outflux of processed information. This notion of streaming data through an AU is crucial to our model. In fact every active AU is constantly receiving an influx of input information and generating an outflux of new information. The output flow may be a simple functional transform of the input or it may be a set of control signals used to affect other AUs.

Unlike tonic AUs, phasic AUs are not always active. They only become active when signaled to do so by other AUs. Phasic AUs are typically used to implement specific behavioral tasks. For example, a phasic AU might encode knowledge needed

to perform a routine task such as scratching an itch, or picking up an object, or walking. An AU doesn't necessarily encode all the knowledge needed to perform a task. Normally an AU will only encode the knowledge needed to perform the task in terms of other AUs. For example, an AU for picking up an object may rely on other subordinate AUs for reaching, grasping, and lifting to perform its task; it doesn't need to know the details of how to grasp but only when and what to grasp.

Thus, a phasic AU is analogous to a function or a procedure in an imperative programming language. Just as procedures are units of abstraction in imperative programming languages, an AU is a unit of abstraction in our model. There is however an important difference. In an imperative language, when one procedure calls another, the caller passes a set of arguments (usually by value) to the callee, who then operates on the arguments (performs some function or some task) and returns a value to caller. While the callee is executing the caller is idle, waiting to be returned to. There is only a single thread of control and it passes from the caller to the callee and then back. In our model when one AU invokes another, the caller streams input to the callee, who in turn processes the input (either computes an output stream or performs some task). In this case, both the caller and the callee are active and data streams through both of them. Effectively the caller is like a flow valve that directs a flow of information (and activation) to the callee. Since the caller continues to be active during the invocation, it can modify the value and even the flow of input data to the callee. The set of AUs that can activate an AU are called its *superiors*. The set of AUs that an AU itself can activate are called its *subordinates*. When an AU activates one of its subordinates it is said to own it. This is because once an AU invokes another (subordinate) AU on its behalf, the subordinate cannot be used by any other AU. If another AU tries to invoke it, a resource conflict arises. It is possible, however, for one AU to preempt an AU from its owner, in which case ownership changes to the new AU.

A phasic AU may become inactive for several reasons. If the owner no longer needs the services provided by the AU, it can signal the AU to deactivate; if the AU finishes the task it was invoked to perform it may deactivate itself; if the AU has not finished its task but is in a position where it cannot proceed, it will signal failure to its owner and deactivate; or, if one invocation of the AU gets preempted by another invocation (via a new caller), the AU will stream a blocked signal to the original owner, reset any internal state (momentarily inactive), and begin processing the new invocation.

6.2.2. Control and Data Streams

What an AU does and how it does it is determined by a set of control productions. Each production is of the form <condition → response>. "Condition" is a predicate on the available inputs and determines when the production should fire. "Response" is the generative half of the production; a response may generate an output value, set local state variables, and send control signals to other (subordinate) AUs. When an AU is active, it is continually evaluating its control productions. With each round of evaluations, the inputs are analyzed, and outputs and control signals are generated. Since changes in the output streams only occur at production firings, the responsiveness of data streams emanating from an AU is directly proportional to the AU's rule evaluation rate. Logically, productions are evaluated instantaneously.

Physically however there are some interesting implementation issues that must be addressed in order to guarantee reasonable semantics and performance.

Each AU has only limited information on which it can base its behavior. This information can be classified into the following logical groups:

1) *Functional Streams.* These are the parameter input and output streams of the AU. They make up the interface specification of the AU as seen by an invoking AU. When an AU is invoked its input streams are driven by the owner. After a short delay, the AU generates the output streams. Functional input streams are commonly used in production conditions. Functional output streams of an AU are less likely to be used by the AU itself. What is more likely is that the owner of the AU will have productions that are contingent on the result in a functional output stream. Under these conditions the owner can monitor the progress of its subordinate AUs and adapt its behavior as appropriate.

2) *Control Streams.* Control streams pass control and synchronization information between interconnected AUs. Each AU has control streams it drives and control streams it monitors. Our model defines seven control signals: ACTIVATE, DEACTIVATE, PRIORITY, ACTIVE, BLOCKED, OUTPUT, and FAILED.

3) *Global Streams.* Global Streams correspond to global variables in imperative programming languages. Global streams provide information to an AU that is not naturally part of the AU's invocation interface. This includes information about certain aspects of the AU's task that the invoking AU would not necessarily know. Tonic AUs are the sources of all global streams.

4) *Subordinate AUs.* As a procedure may call subroutines to perform common functions and tasks in an imperative programming language, so can an AU may invoke a subordinate AU. The output streams of these subordinates are available to the invoking AU's control productions. They may provide results of a functional computation or status information about the sub-AUs progress.

5) *Local Context.* Each AU is also capable of maintaining a local context. This corresponds to local variables found in imperative programming languages. Each time an AU is activated its local context is reset. Control productions are the only means of setting values in the local context. If control of an AU changes from one owner to another, the context of the AU is reset and the original context is lost.

An AU is a resource. It performs a specific task or manages a specific functionality. If two or more superior AUs try to use the same subordinate AU simultaniously a resource conflict arises. In our model, these conflicts are resolved by a simple but powerful priority based scheme. Associated with each AU invocation is a priority value that is determined by the calling AU. When a conflict arises, the request with the highest priority value is given ownership of the resource. High priority requests preempt lower priority request. Invocation priorities are not fixed, but are computed as a function of the current situation. Further, since an AU's priority signal is available in its context, it can compute the priority of a sub-AU's invocation in terms of its own. This leads to a flexible distributed control mechanism

since high priority activities can propagate their priorities to arbitrary layers of subordinate AUs.

So far in this research we have defined the semantics our model precisely; we have chosen and precisely defined a robot domain that will serve as our experimental testbed and completed a preliminary design of its control system; and we have generated the syntax for a programming language that will embody our model.

6.2.3. Future Directions

From this point, we intend to pursue three main lines of research. The first is to implement the programming language that embodies the model. As mentioned the syntax and semantic have been defined, but we have not yet started compiler (or interpreter) construction. The compiler will be designed to generate an intermediate level language that can be interpreted by a uniprocessor (SUN or Lisp machine). Such an interpreter will allow us to study and debug our systems using simulators. A compiler/loader for the intermediate level language will also be built to run under Psyche [Scott et al. 1988] on the BBN Butterfly Multiprocessor.

In parallel with language implementation we intend to develop and build the control system for a mobile robot. The robot's domain is dungeons and dragons like, consisting of a network of rooms and corridors filled with objects and creatures. The purpose of the robot is to survive; that is, to avoid injury and to find sufficient food. The robot will initially be built and debugged on a simulator. As language tools progress, the control system will be implemented on the Rochester Robot.

Finally, our model for robot control is far from complete. (For example, it currently has no mechanism for doing complex reasoning, anticipation, or learning.) The experience we gain implementing a language and our robot control system will provide invaluble information about the shortcomings of our current model and insights into how the model can be extended to overcome those shortcomings.

7. Neural Net Computational Models

Value units are a general way of representing different kinds of *multi-dimensional variables and functions* without requiring that each unit have a large bandwidth. The most optimistic estimate of discriminable signal range in a single neural output is about 1:1000, and this is insufficient to handle multi-dimensional variables. Value units overcome this limitation on neural output by breaking the ranges of a variable up into intervals and representing each interval with a separate unit. These intervals can be organized in many different ways. One straightforward way is to represent a variable $\mathbf{v} = (v_1, ..., v_k)$ isotropically by allocating a unit for each of N^k discrete values. These values are the center of intervals of width $\Delta\mathbf{v} = (\Delta v_1, ..., \Delta v_k)$. The value k is the *dimensionality* of the variable. Synonymously, we will sometimes use the term parameter instead of variable. Two examples of value encodings are shown in Figure 14. Figure 14A shows a highly stylized representation of the orientation-sensitive cortical neurons found in striate cortex. In this case the variable is three-dimensional, with $\mathbf{v} = (x, y, \theta)$. Each unit represents a specific value of (x_0, y_0, θ_0) and has an associated $(\Delta x, \Delta y, \Delta\theta)$ that may be loosely thought of as its receptive field. Figure 14B shows a value unit representation for unit

directions in three-dimensional space. In general, the intervals of neighboring value units will be overlapping.

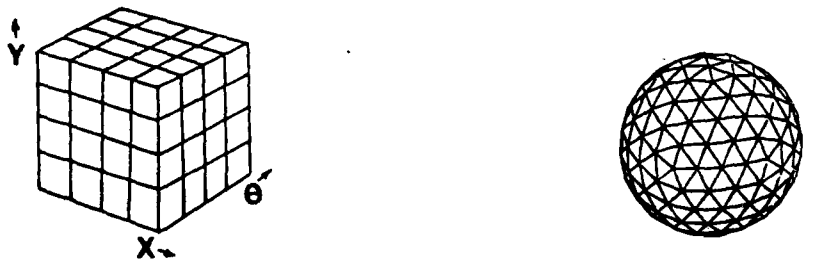


Figure 14: Two examples illustrating the discreteness of value unit encoding. (A) 3-D x-y-orientation units for striate neurons. In this scheme retinotopic coordinates and orientation are regarded as a 3-D space that is covered by the receptive fields of units. (Note that this space is actually toroidal, owing to the periodicity of θ.) (B) 2-D units for indicating directions in space. Directions can be described by coordinates on the unit sphere. The sphere can be tesselated (covered) with uniform intervals. The figure shows a triangular tesselation based on subdividing an icosahedron. If one imagines the figure as a kind of helmet, then the units can indicate directions in space with respect to the head.

Networks of value units can be designed, that are impervious to the loss of single units by: (1) distributing the function among small groups of local units; and (2) allowing for the recruitment [Feldman 1981] of new, previously una'located units.

7.1. Interpolation in Overlapping Receptive Fields

The position paper [Ballard 1987a] elaborates on the basic trade-offs that are entailed by a value unit representation. Basically, the value unit strategy is potentially expensive in the number of units. Two strategies for saving units are *subspaces* and *overlapping receptive fields*. These are diagrammed in Figure 15.

Our result concerns the use of overlapping receptive fields to obtain high precision information. The potential use of receptive fields in this manner has long been suspected. Our model makes the necessary connections with the Hopfield model as well as demonstrating many useful formal properties. The complete result appears in [Ballard 1987c].

Our solution uses a form of Lagrangian interpolation [Davis 1963; Jaeger and Starfield 1974]. Instead of a weight unity for a single discrete value, weighted connections are each of 2^k neighboring values. (There is a way to use only k neighbors using non-orthogonal sampling [Ballard 1987c]; however, the desired properties are more easily understood using orthogonal sampling.) The weights are chosen so that the weighted sum of the discrete values equals the original value. Figure 16 compares the two schemes for the two-dimensional case, i.e., $x = (x_1, x_2)$.

This scheme has the following useful properties, which are developed formally in [Ballard 1987c], but the three most important are:

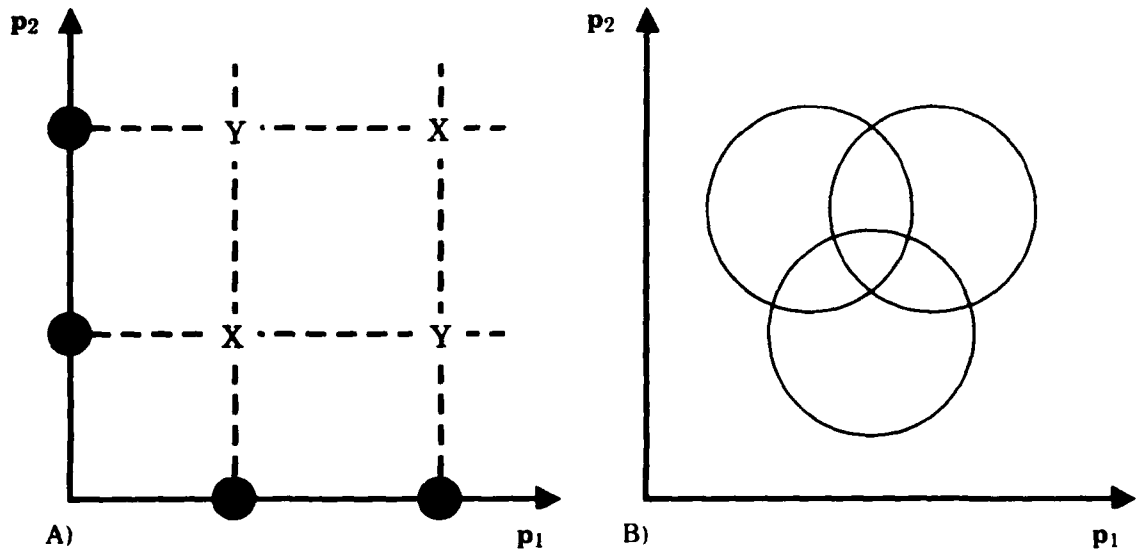


Figure 15: Two ways of reducing the number of value units required to encode a stimulus. (A) Subspaces For a multi-dimensional stimulus $\mathbf{p} = (\mathbf{p}_1, \mathbf{p}_2)$, use units to encode the subspaces $\mathbf{p}_1$ and $\mathbf{p}_2$ separately. This has the disadvantage that for multiple stimuli the possibilities denoted by X cannot be distinguished from Y. (B) Large receptive fields in the high-dimensional space. This strategy also has problems with multiple, closely-spaced stimuli

1) It is insensitive to random measurement noise. If the measurements are corrupted by noise of mean zero then the expected interpolated value is the same as the original value.
2) It can be generalized to parameter spaces of any dimensionality.
3) If the mappings are locally linear over the grid size, i.e., have a Taylor series approximation that requires only first order terms, then the technique can be extended to handle the general case of constraints between networks using different parameters.

One important consequence of this scheme is that the weights can no longer be symmetric. At present, this means that there is no formal convergence theorem, but two ameliorating points are: (a) such a proof may be found for the special hierarchical architectures that we are considering; and (b) a limited number of simulations using nonsymmetric weights in such systems (see next section) have demonstrated them to be stable.

7.2. Learning in Hierarchical Neural Networks

In [Ballard 1987d], we showed that multiple-layer hierarchical systems can be built without necessarily paying a penalty in terms of the convergence time. To do this it was shown that the backpropagation algorithm, when cast in terms of an autoassociative, three-layer network, could be made into a modular system where modules at different levels of abstraction could be coupled. Almost all of the results

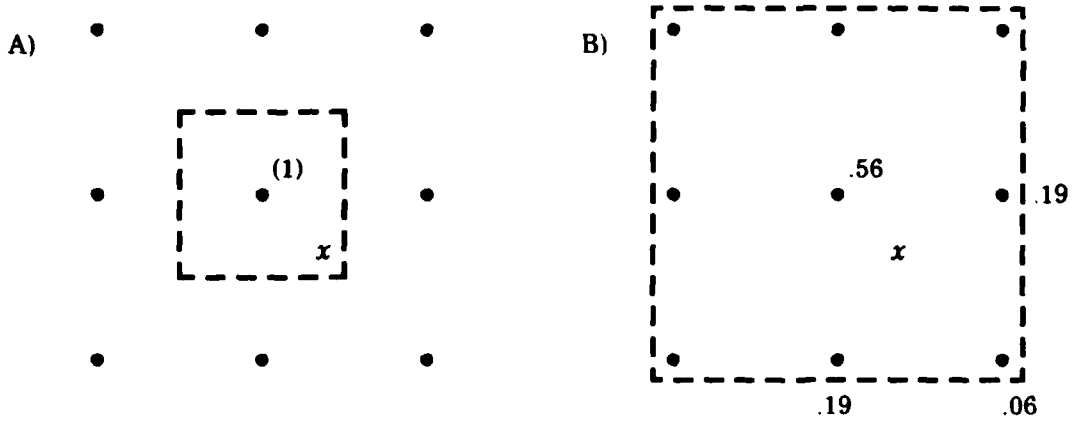


Figure 16: The core problem is one of estimating precise numerical values from discrete samples. A previous method for doing this used a truncation scheme (A) [Duda and Hart, 1972; Ballard, 1984a; Li et al., 1985]. However, truncation is sensitive to the grain size of the quantization. The value can only be reported to within plus or minus one grid unit. Furthermore, the following dilemma occurs. If the grain size is small to minimize the effects of roundoff error, then the individual measurements do not fall in the same quantized cell; whereas if the grain size is large, the roundoff error prohibits accurate localization. Our interpolation method of value encoding is shown in (B). In the new method, receptive fields overlap and the value is distributed among neighboring units. Also, truncation errors are eliminated: in (A) the original value cannot be recovered, whereas in (B) it can

are based on empirical tests, although there are some analytical arguments to suggest that the experimental results should have been expected. The experimental results were very encouraging. The conventional backpropagation architecture, when extended to three internal levels instead of one, did not converge, whereas the comparable modular architecture (with twice as many connections, however) converged in a time comparable to that of the much smaller system with only one internal layer. However, an important disadvantage of this scheme arises in the use of the network. In the case where a sensory input is provided and a motor response is desired, only one-half of the inputs to the abstract layer will be present. This placed a much greater demand on the abstract layer to determine the correct pattern instead of an alternate. In experiments, many patterns could not be completed.

To fix this problem, we have adopted a new recursive backpropagation scheme by Pineda [1987] and extended it significantly. In addition, Patrice Simard from our laboratory has studied a small feedforward network that is used to solve the parity problem (Figure 17).

Sixteen patterns on four input units are presented and the network is trained using both the [Rumelhart et al. 1986] algorithm and the new algorithm. Figure 18 (from [Simard et al. 1988]) shows that the standard algorithm is easily trapped in local minima and does not learn large fractions of the training set of patterns. In contrast, Simard's use of simulated annealing instead of the usual gradient search to change the network weights has yielded strikingly better results. In the simulation run shown the learned patterns went from 40% to 100%.

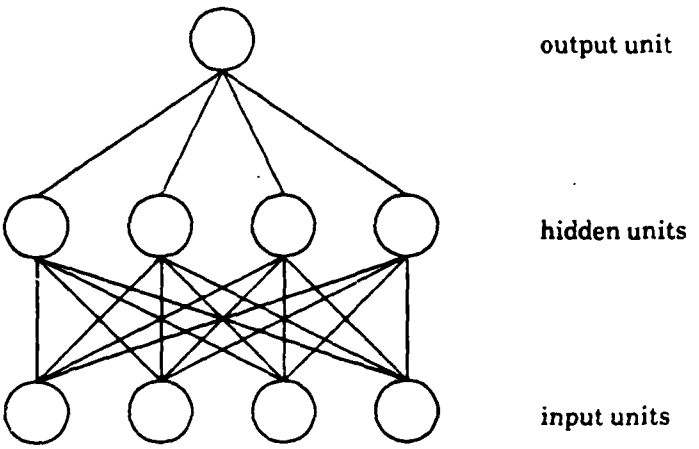


Figure 17: Feedforward network used in simulated annealing study.

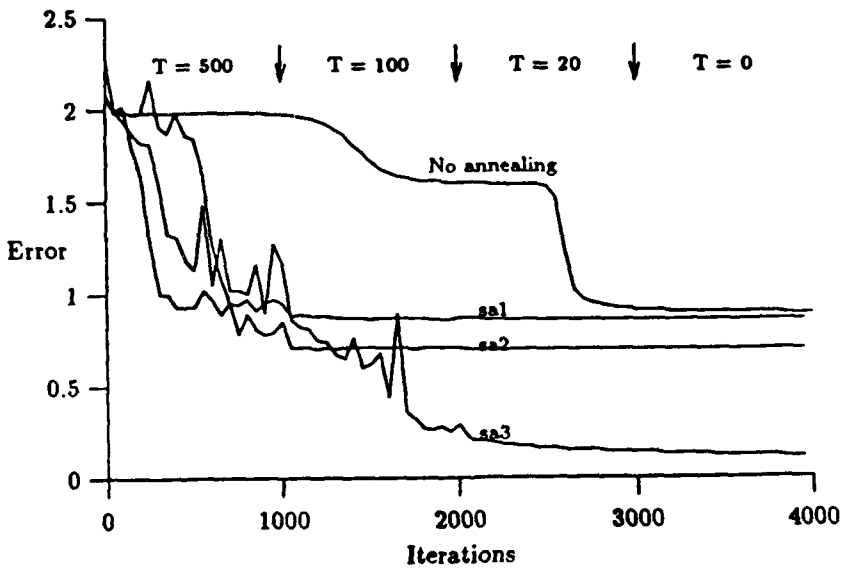


Figure 18: Learning the parity problem with simulated annealing. Here sa1, sa2, and sa3 correspond to three different runs with different initial weights using the simulated annealing with temperatures of 500, 100, 20, and 0.

8. Utility and Pilot Projects

Many capabilities remain to be implemented and incorporated into our research, and many ideas remain to be tried. Many of the following isolated projects are mentioned above as integral parts of larger projects. At the risk of some redundancy they are listed here as independent projects.

1) *Simultaneous Head and Eye Movement.* Investigate the performance of the system when simultaneous head and eye movements are commanded. It is important to know how to, and how well we can, synchronize these movements.

2) *Which Binocular Stereo Algorithm?* We currently have two simple algorithms for deriving a point's 3-D location from a set of 2-D image points corresponding to it. Is there a reason to prefer one over the other? Criteria for a judgement might be extensibility to more than two images, accuracy, consistently quantifiable errors (or residuals) for use in discarding bad data points.

3) *Motion-Derived 3-D Location.* Use multiple images from different locations with simple extensions of the current binocular stereo code to improve 3-D location accuracy, or to generate 3-D locations from motion instead of binocularity. This capability interacts well with tracking: from different points of view an object may look significantly different, but if it has been tracked its identity is assured, and it may be located in 3-D with confidence.

4) *Integrate Robot Lab with MIMD Computation.* The Psyche project is dedicated to early integration with vision and robot applications. A first project might be to keep the database of object characteristics and locations on the B+P. This application is within the capabilities of the initial serial line interface.

5) *Jacobian Calculations.* Derive the differential motion relationships between head and eyes by standard techniques as a first step in continuous hand-eye coordination [Paul 1981].

6) *Moving Objects.* The model train can be controlled remotely, and seems currently to be the easiest way for us to generate controlled object motion. It has the disadvantage of providing essentially 2-D motion, but the advantage of controllable (and slow) speeds, and of repeatable and measurable trajectories.

7) *Trajectory Calculations.* Given several 3-D points acquired through time, and assuming they arise from a moving rigid object, how should the trajectory be calculated and described? Issues of timestamping arise here. Clearly there is no "right" answer for trajectory-characterization, but what is one useful answer?

8) *Multiple Object Management.* Keeping track of multiple objects in a fast vision system raises several important issues. The current stereo implementations only attempt to estimate a 3-D location when there is exactly one blob in each image. Currently, we check the error of the estimated location and discard the estimate if the error is too large. Multiple objects can be handled by several means: the epipolar constraint limits the location of corresponding points in two images to a 1-D locus that is a function of camera location. Relational structures of objects, enhanced with object characteristics, can be matched between images by graph matching techniques to establish correspondences. Consistent 3-D locations or

trajectories can be used to verify correct object matches in a static or dynamic environment.

9) *Environment Mapping.* The current object-detection algorithm embodied in the MaxVideo can be made more robust (mainly by extending the domain of objects beyond that of light features against dark backgrounds). The existing BMW exploration code can then be expanded to create a 3-D map of interesting objects in the robot's enviroment. The appeal of such a project depends more on its style and methods than on its end product. The "explore" capability might be a basic building block in a hierarchy of behaviors, for instance [Brooks 1987].

10) *VOR and Segmentation.* The ability to pursue a moving object smoothly is basic to our human ability to recognize or describe the object. Likewise, the ability to examine an object from a continuum of points of view is a powerful aid to recognition. The latter capability can be realized by mechanisms that hold eye fixation over head movements, such as object tracking and the VOR. This pilot project is to investigate the value added to the segmentation and recognition problem by multiple viewpoints. In particular the continuous version might maintain a cumulative, time-averaged or moving-average image created in MaxVideo. The expectation is that non- fixated objects will blur, and the fixated one will be visually easily discriminable.

11) *Vision Operators.* A useful contribution would be a library of templates for the real-time correlation board, VFIR-II. Since VFIR can hold 128 such templates, they could make up part of the generic system setup sequence. There are several "interest operators" in the literature [Bandopadhay 1986; Morovec 1977; Thorpe 1983] that can help object-tracking code decide which points on an object will be easy to track. These non-linear operators are not directly implementable in VFIR but might make an easy and useful Euclid capability.

12) *Explore Hardware Limits.* In the context of another project it would be useful get increased data bandwidth by passing image data to the Euclid computer over the MaxBus rather than through the Sun and the VMEBus. On another front, the FPP-BPP with its VMEbus connection could be used for (camera) motor control or other distributed operations.

13) *Learning.* The control of multiple degree-of-freedom systems is of continuing intellectual and practical interest. One recent line of work involves learning in parallel neural-net architectures [Jordan 1988]. Our own work in back-propagation learning [Simard et al. 1988] makes experimentation with such an approach easy. Candidate relations for learning include camera calibration, the Jacobian relationships of the robot and head joint angles, and the inverse kinematic relationship between a point to be fixated, head location, and the camera altitude and azimuth angles. Automated learning of general relationships can avoid the problems of a priori but mistaken imposition of specific relationships upon a system (e.g., our pinhole camera assumption). Further, certain types of learning are known to yield significant speedups in acquiring practical competence [Jordan 1988]. Bearing in mind that some successful builders of real-time robotic systems think learning should be used only with discretion [e.g., Andersson 1988], it

seems inevitable that automatic and repeatable adjustment, correction, and optimization of parameters, and the automation of skill-formation and retention, must form an important part of what we consider an active intelligence.

9. References

Accetta, M., R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, and M. Young, "Mach: A new kernel foundation for Unix development," *Proc., Summer 1986 Usenix Conf.*, 1986.

Agre, P.E. and D. Chapman, "Pengi: An implementation of a theory of activity," *Proc., AAAI87*, vol. 5, pp. 268-272, 1987.

Allen, J.F. *Natural Language Understanding.* Menlo Park, CA: Benjamin-Cummings, 1987.

Allen, J.F. and J. Koomen, "Planning using a temporal world model," *Proc., 8th Int'l. Joint Conf. on Artificial Intelligence*, pp. 741-747, 1983.

Aloimonos, J., I. Weiss, and A. Bandopadhay, "Active vision," *Proc., 1st Int'l. Conf. on Computer Vision*, pp. 35-54, June 1987.

Andersson, R.L. *A Robot Ping-Pong Player: Experiment in Real-Time Intelligent Control.* Cambridge, MA: MIT Press, 1988.

Annaratone, M., E. Arnould, T. Gross, H.T. Kung, M.S. Lam, O. Menzilcioglu, K.Sarocky, and J.A. Webb, "Warp architecture and implementation," *Proc., 13th Annual IEEE/ACM Int'l. Symp. on Computer Architecture*, pp. 346-356, June 1986.

Ballard, D.H., "Cortical connections: Structure and function," *Behavioral and Brain Sciences 9*, 1, 67-120, March 1986; also in M. Arbib and A. Hanson (Eds). *Vision, Brain and Cooperative Computation.* Cambridge, MA: MIT Press/Bradford Books, 1987a.

Ballard, D.H., "Eye movements and spatial cognition," TR 218, Computer Science Dept., U. Rochester, November 1987b.

Ballard, D.H., "Interpolation coding: A representation for numbers in neural models," TR 175 (revised), Computer Science Dept., U. Rochester, September 1986; *Biological Cybernetics 57*, pp. 389-402, 1987c.

Ballard, D.H., "Modular learning in neural networks," *Proc., Nat'l. Conf. on Artificial Intelligence (AAAI)*, July 1987; TR, Institute for Theoretical Physics, U. California, Santa Barbara, January 1987d.

Ballard, D.H. and A. Ozcandarli, "Eye movements and visual cognition: Kinetic depth," *Proc., Int'l. Conf. on Computer Vision*, December 1988.

Bandopadhay, A., "A computational study of rigid motion perception," TR 221 and Ph.D. Thesis, Computer Science Dept., U. Rochester, December 1986.

Brooks, R.A., "Intelligence without representation," *Proc., Workshop on Foundations of Artificial Intelligence*, pp. 1-21, 1987.

Brown, C.M., "Parallel vision with the Butterfly computer," *Proc., 3rd Int'l. Conf. on Supercomputing*, Boston, May 1988.

Brown, C.M., D.H. Ballard, and O.A. Kimball, "Constraint interaction in shape-from-shading algorithms," *1982-83 Computer Science and Engineering Research Review*, Computer Science Dept., U. Rochester, Fall 1982.

Brown, C.M. and R.D. Rimey, "Coordinates, conversions, and kinematics for the Rochester Robotics Lab," TR 259, Computer Science Dept., U. Rochester, August 1988.

Burt, P.J. and G.S. van der Wal, "Iconic image analysis with the pyramid vision machine (PVM)," *Proc., 1987 IEEE Workshop on Computer Architecture for Pattern Analysis and Machine Intelligence*, pp. 137-144, October 1987.

Chou, P.B. and C.M. Brown, "Multimodal reconstruction and segmentation with Markov Random Fields and HCF optimization," *Proc., DARPA Image Understanding Workshop*, April 1988; submitted, *Int'l. J. Computer Vision.*

Coombs, D., "Spot, a new-age explorer," Programming Project Report, Computer Science Dept., U. Rochester, 1988.

Coombs, D.J. and B.D. Marsh, "ROVER: A prototype active vision system," TR 219, Computer Science Dept., U. Rochester, August 1987.

Daniel, R.W., M.A. Irving, A.R. Fraser, and M. Lambert, "The control of compliant manipulator arms," in R.C. Boles and R. Roth (Eds). *Robotics Research (4th Int'l. Symp.).* Cambridge, MA: MIT Press, pp. 119-125, 1988.

Davis, P.J. *Interpolation and Approximation.* Blaisdell Publishing Co., 1963.

Digital Equipment Corporation, DECnet Digital Network Architecture, Digital Data Communications Message Protocol (DDCMP) Specification, Version 4.0, 1978.

Duda, R. and P. Hart. *Pattern Classification and Scene Analysis.* New York: John-Wiley and Sons, 1973.

Feldman, J.A., "Four frames suffice: A provisional model of space and time," *The Behavioral and Brain Sciences 8*, pp. 265-289, 1985.

Feldman, J.A., "Memory and change in connection networks," TR 96, Computer Science Dept., U. Rochester, October 1981.

Feldman, J.A. and R. Sproull, "Decision theory and artificial intelligence II: The hungry monkey," *Cognitive Science 1*, pp. 158-192, 1977.

Feldman, J.A. and Y. Yakimovsky, "Decision theory and artificial intelligence I: A semantics-based region analyzer," *Artificial Intelligence 5*, pp. 349-371, 1974.

Firby, R.J., "An investigation into reactive planning in complex domains," *Proc., AAAI*, vol. 5, pp. 202-206, 1987.

Fowler, R.J., T.J. LeBlanc, and J.M. Mellor-Crummey, "An integrated approach to parallel program debugging and performance analysis on large-scale multiprocessors", *Proc., ACM Workshop on Parallel and Distributed Debugging*, pp. 163-173, May 1988.

Franklin, G.F. and J.D. Powell. *Digital Control of Dynamic Systems.* New York: Addison-Wesley, 1981.

Georgeff, M.P. and A.L. Lansky, "Reactive reasoning and planning," *Proc., AAAI*, vol. 5, pp. 677-682, 1987.

Goldenberg, R., W.C. Lau, A. She, and A.M. Waxman, "Progress on the prototype PIPE," *Proc., IEEE Workshop on Computer Architectures for Pattern Analysis and Machine Intelligence*, pp. 67-74, October 1987.

Hanson, A. and E. Riseman, "The VISIONS image-understanding system," in C.M. Brown (Ed). *Advances in Computer Vision.* Hillsdale, NJ: Lawrence Erlbaum Assoc., Pub., pp. 1-114, 1988.

Hayward, V., "RCCL User's Guide," Computer Vision and Robotics Lab., McGill Res. Centre for Intelligent Machines, McGill U., April 1984.

Hein, A. and M. Jeannerod. *Spatially Oriented Behavior.* New York: Springer-Verlag, 1983.

Jaeger, J.C. and A.M. Starfield. *An Introduction to Applied Mathematics.* Oxford Press, 1974.

Jordan, M., "Supervised learning and systems with excess degrees of freedom," COINS TR 88-27, U. Massachusetts, May 1988.

Kautz, H.A., "A formal theory of plan recognition," TR 215 and Ph.D. thesis, Computer Science Dept., U. Rochester, May 1987.

Kuo, B.C. *Automatic Control Systems.* Prentice-Hall, 1982.

LeBlanc, T.J., M.L. Scott, C.M. Brown, "Large-scale parallel programming: Experience with the BBN Butterfly Parallel Processor," *Proc., ACM SIGPLAN PPEALS*, pp. 161-172, July 1988.

Little, J.L., G. Blelloch, and T. Cass, "How to program the connection machine for computer vision," *Proc., IEEE Workshop on Computer Architectures for Pattern Analysis and Machine Intelligence*, pp. 11-18, October 1987.

Miles, F.A., "Adaptive regulation in the vergence and accommodation control systems,", in A. Berthoz and G.M. Jones (Eds). *Adaptive Mechanisms in Gaze Control: Facts and Theories.* Elsevier, 1985.

Miles, F.A., L.M. Optican, and S.G. Lisberger, "An adaptive equalizer model of the primate vestibulo-ocular reflex," in A. Berthoz and G.M. Jones (Eds). *Adaptive Mechanisms in Gaze Control: Facts and Theories.* Elsevier, pp. 313-326, 1985.

Moravec, H.P., "Towards automatic visual obstacle avoidance," *Proc., 5th IJCAI*, p. 584, August 1977.

Paul, R.P. *Robot Manipulators.* Cambridge, MA: MIT Press, 1981.

Paul, R.P., J.Y.S. Luh, S.Y. Nof, and V. Hayward, "Advanced industrial robot control systems," TR-84-25, School of Electrical and Industrial Engg., Purdue U., July 1984.

Pellionisz, A., "Tensorial aspects of the multidimensional approach to the vestibulo-oculomotor reflex and gaze," in A. Berthoz and G.M. Jones (Eds). *Adaptive Mechanisms in Gaze Control: Facts and Theories.* Elsevier, pp. 281-296, 1985.

Pineda, F.J., "Generalization of backpropagation to recurrent and high-order networks," *Proc., IEEE Conf. on Neural Information Processing Systems--Natural and Synthetic*, November 1987.

Rumelhart, D.E., G.E. Hinton, and R.J. Williams, "Learning internal representations by error propagation," in D.E. Rumelhart and J.L. McClelland (Eds). *Parallel Distributed Processing.* Cambridge, MA: MIT Press/Bradford Books, pp. 318-364, 1986.

Scott, M.L., T.J. LeBlanc, and B.D. Marsh, "Design rationale for Psyche, a general-purpose multiprocessor operating system," *Proc., 1988 Int'l. Conf. on Parallel Processing*, August 1988; to appear, *Computer Science and Engineering Research Review*, Computer Science Dept., U. Rochester, September 1988.

Shafer, S.A., "Automation and calibration for robot vision systems," TR CMU-CS-88-147, Computer Science Dept., Carnegie- Mellon U., May 1988.

Simard, P.Y., M. Ottaway, and D.H. Ballard, "Analysis of recurrent back-propagation," *Snowbird Conf. on Neural Networks*, April 1988; *Proc., Carnegie Mellon U. Connectionist Models Summer School*, June 1988; submitted, *Neural Inf. Processing Systems.*

Terzopolous, D., "Multilevel computational processes for visual surface reconstruction," *Computer Vision, Graphics and Image Processing 24*, 1, 52-96, 1983.

Thorpe, C.E., "An analysis of interest operators for FIDO," TR 83-19, Robotics Inst., Carnegie Mellon U., December 1983.

Timoshenko, S. *The Theory of Elastic Stability.* New York: McGraw-Hill, §2.7, pp. 77-79, 1961.

Unimation Incorporated. *User's Guide to VAL II.* Danbury, CT, 1986a.

Unimation Incorporated. *Unimate PUMA Mark III Robot 700 Series Equipment Manual.* Danbury, CT, 1986b.

Vere, S., "Planning in time: Windows and durations for activities and goals," *IEEE Trans. PAMI 5*, 3, pp. 246-267, 1983.

Yeshurun, Y. and E.L. Schwartz, "Cepstral filtering on a columnar image architecture: A fast algorithm for binocular stereo segmentation," Robotics Research TR 286, Courant Inst. of Mathematical Sciences, New York U., March 1987.

10. Appendix 1: Programming a Real-Time Image Pipeline Processor

In the vision research community there is a much current interest in performing early vision by special purpose image processing machines. It is now possible to perform operations such as image filtering, convolution, thresholding, histogramming, etc. in hardware. Several different systems have been developed to do these types of calculations, such as the PIPE system [Goldenberg et al. 1987] (references are found with those of the main body of the paper), the WARP computer [Annaratone et al. 1986], the Pyramid Vision Machine [Burt and van der Wal 1987], and the MaxVideo System. It is interesting to note the Connection Machine at M.I.T. has been used for these purposes as well [Little et al. 1987]. The advantage of these systems is that they can process images at essentially frame rates and can then convert the image data into symbolic data that can be processed by more general purpose computers. This conversion into symbolic data effectively reduces the amount of data, to a level that can be handled by general-purpose computers. This gives these computers a higher level of abstraction.

At the University of Rochester we have an interest in parallel architectures as well as an interest in early vision. For this reason we are interested in image processing hardware that can be scaled up in parallel much like our Butterfly computer. We have chosen the MaxVideo system from DataCube.

10.1. MaxVideo

The MaxVideo system is a set of image processing boards that plug into a VME-BUS chassis. One can create the system modularly by using a set of boards compatible with the needs of the system. The boards are connected with ribbon cables which act as image busses that pipe the images around the system at frame rates. Different boards can be processing image data at the same time. This means that if there are three convolution boards in the system, then three convolutions can be performed in

parallel. System upgrading is easy and fairly inexpensive in that one need only add the boards desired to the VME-BUS cage.

The disadvantages to the MaxVideo system are the large learning curve needed to program the system, the lack of portability of code from one application to another, and the general difficulty of programming the system even for one who understands the system. The lack of portability of code stems from the fact that people write applications by cabling the boards with ribbon cables and writing code to push images around the system through those cables. The code is therefore dependent on the configuration of the ribbon cables. Another problem associated with the programming of these boards is the timing of the image pipelines. As the image flows from section to section of each board it is delayed by various sections. This delay is based on the function being performed as well as the size of the image. This makes it very difficult to program. One must know how to calculate these delays for each section of each board. A third problem is the interactions that occur between boards, which makes the state of the entire system very delicately dependent on the state of each of its components.

10.2. PIPE

The PIPE system has a smooth and accessible user interface, which capitalizes on certain performance tradeoffs. The PIPE system is very easy to program. It has a graphical interface that allows one to program the machine to do low level image processing (image subtraction, filtering, etc.) without writing any code. Through the use of this interface one creates sets of large microcode instructions which are downloaded to the actual PIPE machine. The machine is a collection of identical processing stages. This uniformity of stages makes the machine easier to program but reduces flexibility. As the image moves out of one stage it moves into another. Each stage has image buffers. At the end of each stage the results are stored into a buffer. This keeps the programmer from worrying about timing considerations because there is a 60th of a second time clock and the images are read from memory, processed and stored in that time period. It may take several stages or several repetitions through a stage to accomplish the desired effect. To add processing power, one merely adds another stage. The problem with this is that to add processing hardware you must add it to the end of the pipeline and thus add it in time. There is no way to have several pipelines in parallel. This is a serious draw back. In short, the architecture of the PIPE did not suit our computational needs but was a beautiful model of how the programmer could accomplish many tasks quickly (as regards programmer time) and easily. What might take man weeks to program the MaxVideo system would take man days on the PIPE system.

Another relevant effort is the interface package MAGICAL, which provides interactive, but not real time, access to a MaxVideo configuration. This system was implemented as C library for doing many standard functions (initialization, image loading and saving to disk, standard point and neighborhood operations, etc.) and is accompanied by a windowed menu driven interface for calling these functions from the shell, called WAND. This system used one standard hardware configuration.

10.3. General Needs

Our needs can be summarized as follows.

1) Our emphasis is on real-time image understanding and as such we need to implement system software that will put in these real-time constraints. In addition to this, one or more abstraction tools are needed to help users program more efficiently and hopefully to help with identifying problems in the correctness of algorithms.
2) The system should allow for an evolving hardware configuration. This would require that the configuration be specified in tables upon which the system software would be based. Graphical tools should be provided for creating this configuration which would let the user interactively create a collection of boards and design the connections between them. This program would then generate the tables.
3) The system should allow the concept of parallel hardware units used by cooperating processes. These units can be a collection of any number of boards. The system should also allow the time sharing of these units between cooperating processes.
4) In MaxVideo timing is very important. To perform an operation on two images such as addition or subtraction one must insure that both images get to the correct unit at the exactly the same time. Identifying and quantifying all of the timing delays occur in the system and what amount of delay there is at each point is not an easy task. A tool for helping these calculations would be very useful.

Our goal is to make the MaxVideo system easier to use without losing the flexibility it offers. The MaxVideo system cannot be made as easy to use as the PIPE because of the extra flexibility that it provides, which inherently demands more of the programmer. However, we believe that through software tools and libraries the MaxVideo system can be made easier to use. Further, we believe a generally useful cabling system can be designed to support many applications.

10.4. Abstraction

The heart of the needs is to make the programming of the system conceptually easier. To do this an abstraction tool is needed. The abstraction tool we have chosen is the Finite State Machine (FSM). It would be useful if this abstraction technique could be used at multiple levels. So we should designate a FSM to be a collection or set of boards. This set can range from one to some finite number of boards N. In all practicality the set will probably not be greater than 32 but that is not a requirement. Given that the FSM is a set of boards then the state of the machine would be represented by the concatenation of the register space of each board in the set. This state shall be called a state-word (SW). The programming of the system would be accomplished by creating a state machine program. Each line of the program would be a state-word. Given a library of states we can write programs such as:

```
/* --- initialize the machine --- */
init = LoadState(init.MVSTATE);   /* load state from library */
ExecuteState(init);   /* write the state to board */

/* --- set up image pyramid system --- */
pyramid = LoadState(pyramid.MVSTATE);   /* load state from library */
ExecuteState(pyramid);   /* write the state to board */
```

This program initializes the system and then sets up the registers so that the system would be creating an image resolution pyramid. Now of course we really have no flow of control here so we would want to be able to execute a state, wait for some event, and then change state.

```
init = LoadState(init.MVSTATE);
acq = LoadState(imageAcquire.MVSTATE);
extract = LoadState(featureExtract.MVSTATE);

ExecuteState(init);   /* write the initialized state to board */
ExecuteState(acq);   /* set state to image acquisition */
Wait(oneImageFrame);   /* wait for the acquisition */
ExecuteState(extract);   /* extract a feature list */
Wait(extComplete);   /* wait for the extraction to complete */
```

The above program acquires an image and extracts a list of pixel coordinates based on brightness values of those pixels. These *xy* coordinates are stored on one of the boards. The program would then after the extraction need to read these points from the board in order to calculate the centroid of the "feature" or "blob" of pixels. A call could be implemented for that as well.

```
FetchData(SourceAddress, DestinationAddress, Length);
```

So far we have considered the case of treating the entire collection of boards as one set and thus one FSM. But of course we might want to break the hardware up into a number of different state machines. These machines may or may not be identical. This requires a change in the syntax to allow specification of which FSM is being addressed. This can be done easily by adding a FSMID to each call. In the example consider the state machine board set to contain all of the boards and call the identifier "wholeBunch." The board set information will be contained in a file "wholeBunch.FSMID." To create the identifier one opens the state machine by reading this information from the file and passing a pointer to this information back from the open call.

```
Main()
{
    FSMid wholeBunch;

    StateWord init, acq, extract;
    int xyList[1000];
    int xyCount, Xcenter, Ycenter;
```

```
    wholeBunch = SMOpen("wholeBunch.FSMID");

    init = LoadState(wholeBunch, init.MVSTATE);
    acq = LoadState(imageAcquire.MVSTATE);
    extract = LoadState(featureExtract.MVSTATE);

    ExecuteState(wholeBunch, init);   /* write the init state to machine */
    ExecuteState(wholeBunch, acq);   /* set state to image acquisition */
    Wait(wholeBunch, oneImageFrame);   /* wait for the acquisition */
    ExecuteState(wholeBunch, extract);   /* extract afeature list */
    Wait(wholeBunch, extComplete);   /* wait for extraction to complete */

    /* --- get the list of coordinates and the coordinate count --- */
    FetchData(wholeBunch, FM←XYListAddr, xyList,FM←XYLIST←LENGTH);
    FetchData(wholeBunch, FM←XYCountAddr, xyCount,FM←XYCOUNT←LENGTH);

    /* --- calculate the centroid of the extracted feature --- */
    CalcCentroid(xyList, xyCount, &Xcenter, &Ycenter);
}
```

It is likely that the syntax will change after design and prototyping have been done but one can get a glimpse of idea from the above description. It becomes obvious that creating state-words by hand would not be a pleasant task. Looking up all the register definitions would be crude and laborious. Thus tools are needed to help in this endeavor. These tools should be graphical in nature. One way to do this would be to create a tool for each board that would present a graphical representation of the board. The user could then click on the different sections of the board and upon selecting a section, be prompted for information necessary to program that section of the board. Thus each board editor would produce a state-segment which represents the state of the board. State-words that have a board set larger than one could have custom tools that call the board level tools and concatenate the state-segments in some prescribed manner to create the state-word. FSMs that contain only one board would have state-words that are identical to the state-segment.

The State Machine would be a large benefit in creating a system that is easier to use. First, it provides an abstraction for the system which is easy to understand and well understood in the computing community. Second, it offers this abstraction at many levels from the board level to a full image understanding machine. This provides great flexibility in its use. The lower levels can be used as building blocks to create higher levels of abstraction.

The use of a library of state words allows one to use state-words from other programs and from other programmers as long as they are documented properly. This need motivates a documenting facility that allows one to keep the documentation with the state-word so that by loading a word into the state-word editor, the documentation on that word can be displayed. If one wished the state word can then be manipulated and saved under another name with the documentation.

11. Appendix 2: Controlling Highly Flexible Arms

11.1 Introduction

The goal of this segment of the project is the ultimate flexible robot, one that exhibits large deflections in the technical sense: transverse deflections of the end of a given link comparable to the length of that link. Imagine a fishing pole with a large weight on the end, a "fishing pole robot." This is a model of a robot that is much less massive than its load. The advantages are obvious: the saving of weight and expense. The major disadvantage is that the control problem is apparently very complicated.

Our efforts this summer have been directed at isolating a tractable model with important features of the full fishing pole problem in order to study its dynamics. We expect to build such a device in the near future, but at present we are concentrating on a computer simulation using the correct dynamics of the model. Our model is a simple beam of constant cross-section (it is easy to incorporate a nonconstant cross-section in the model at a later stage), very much wider than it is thick, so that its motion is confined to a plane (we ignore the possibility of twisting at this stage of development). At the end of the beam is a point mass much heavier than the beam.

The beam is supposed to be made of a linear elastic material, and to undergo large deflections within the limits of linear elasticity (small strains). This lets us use the theory of the *elastica*, an invention of the nineteenth century. This is a static theory for the post-buckling behavior of beams, computing beam shapes based on local equilibrium. We extend the elastica to a quasistatic analysis, valid when the speed of deformation is slow compared to the speed of elastic waves in the beam. (The speed of elastic waves in most solids is considerably greater than the speed of sound in air, 300 m/s.)

The quasistatic analysis assumes that the shape of the beam adjusts "instantaneously" to its loading. All the dynamics and control can be put into the forces and torques at the ends. The analysis of a single link would proceed by fixing the coordinate system in one joint of the link, and letting that coordinate system rotate with the joint. The coordinate system is then noninertial, but the boundary conditions on the fixed end become simple, and noninertial effects can be reduced to the usual fictitious forces. Thus, if we can learn to understand the behavior of a beam with a force on it, we can analyze this more complex situation in a relatively straightforward manner. Once the analysis capability is developed, control can be added, and various control strategies can be tested, both in the simulation and in eventual laboratory models.

11.2 Where We Stand

We have an algorithm to solve the static (buckling) problem, and have nearly completed the first dynamic algorithm. The physical principle and its translation into computation are apparently correct; some details of the beam reversal remain to be fixed. We will outline the physics and mathematics of the problem briefly to make it easier to understand our methods.

Consider a uniform beam fixed at one end (see Figure 19). Construct an x,y,z coordinate system such that the fixed end of the beam is at

$$x = 0 = y; \qquad -b/2 < z < b/2$$

and let the undisturbed shape of the beam be along the y axis. Let the beam be of uniform rectangular cross section, height h and width b, and let its length be L. Let $h/b << 1$ so that motion is confined to the x-y plane. The appropriate moment of inertia is $I_{zz} = bh^3/12$. Let the mass of the beam be m, and its Young's modulus be E.

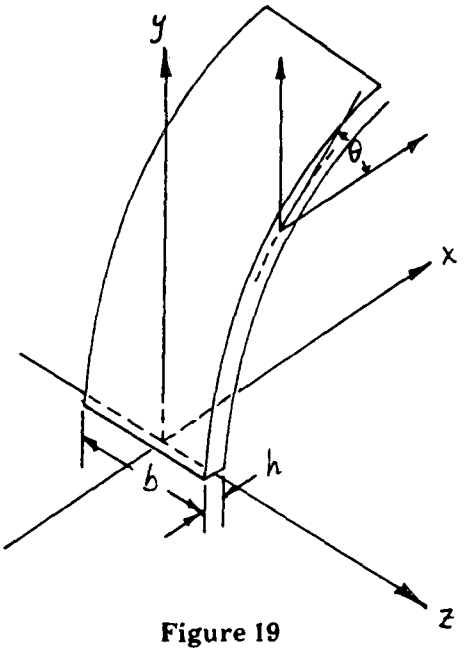


Figure 19

The simplest problem is that of the elastica: a force P in the x-y plane is applied to the free end and the deformation is calculated. It is convenient to scale the problem, letting L be the length scale and $EI\pi/4L^2$ (the critical buckling load for a free-fixed beam) be the force scale. Let s be the dimensionless arc length along the beam, $0 \le s \le 1$, and let $\theta(s)$ denote the local slope of the beam. Let P denote the magnitude of the applied force and ϕ the angle of application, $0 \le \phi \le 2\pi$. The scaled equilibrium equation is then

$$\theta_{ss} = P\sin(\theta - \phi) \qquad (1)$$

where the subscript denotes differentiation. The boundary conditions are $\theta(0) = \pi/2$ and $\theta_s(1) = 0$. This nonlinear problem has implicit solutions in terms of elliptic integrals. These are given in [Timoshenko 1961] for the case $\phi = 3\pi/2$, the classical buckling problem.

The elliptic integrals are not particularly informative, no easier to use than a direct numerical solution, and not adaptable to the eventual dynamic problem, so an algorithm was devised. Equation (1) is integrated from the origin to $s=1$, using a guess for the value of $\theta_s(0)$. The slope at the free end is evaluated, and $\theta_s(0)$ is modified until the slope at the free end is zero. The code was checked against the buckling analysis in Timoshenko, and is satisfactory. Figure 20 shows a sketch of a buckled beam calculated using our method. The scaled force is twice the buckling force, and oriented 45° below the horizontal, as shown in the figure.

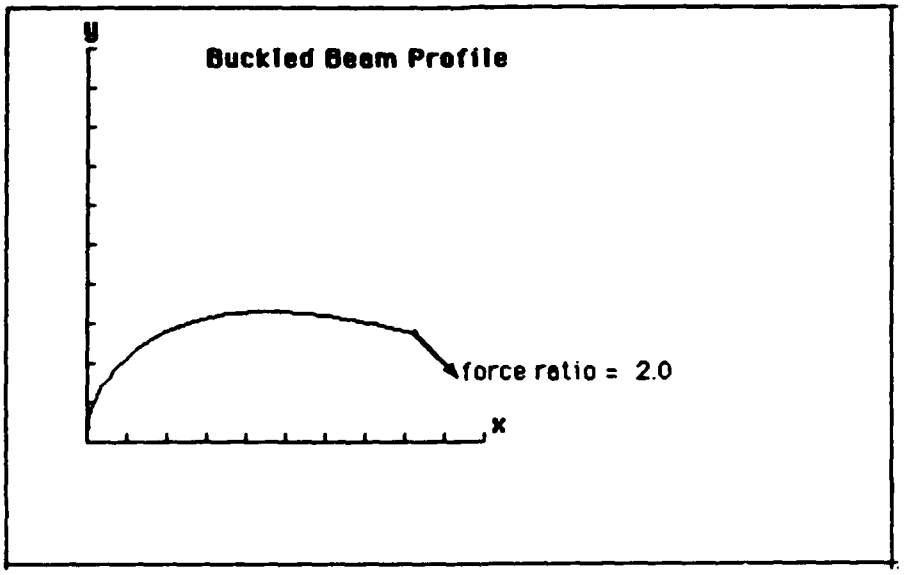


Figure 20

The dynamic problem adds a large mass Mm at the free end. We assume $M >> 1$, so that the mass of the beam can be neglected, and let x^* and y^* denote the x and y coordinates of the mass, located at the end point of the beam. If time is scaled by the time an elastic wave takes to travel the length of the beam, approximately $L\sqrt{[\rho/E]}$ (the shortest conceivable time scale for this problem), then the scaled equations of motion for the mass are

$$M\mathbf{r}_{tt} = P[(\pi h/L)^2/48]\{\mathbf{i}\cos\phi + \mathbf{j}\sin\phi\} \qquad (2)$$

where P and ϕ are defined as above, t denotes time, and $\mathbf{r} = x^*\mathbf{i} + y^*\mathbf{j}$, with $\mathbf{i}, \mathbf{j}$ the Cartesian unit vectors. Equation (2) can be derived from the free body diagram of the mass. The formulation is completed by initial conditions on the displacement and velocity of the end point.

The dynamics problem requires simultaneous solution of (1) and (2). They are coupled by the condition that $\mathbf{r}$ be at the end of the beam:

$$x^* = \int_0^1 \cos\theta ds; \qquad y^* = \int_0^1 \sin\theta ds \tag{3}$$

In principle one should be able to use this relation to eliminate P and ϕ. As it happens, this is not simple, and an alternative method of solution was sought. It is not the only alternative, and others are under active investigation.

We state the following hypothesis: the beam always moves to maximize the change in potential energy. This means that the beam uncoils to decrease potential energy as rapidly as possible, and when it bends, it bends to increase its stored energy as rapidly as possible. We use this hypothesis, and the conservation of energy, to construct an algorithm. First we write expressions for the two energies, under the assumption that $M >> 1$ so that the kinetic energy T is that of the mass alone. The potential energy V is the elastic energy of the deformed beam. Scaling the energy by $bhLE$ leads to scaled energies

$$T = \tfrac{1}{2}M(x^{*2}_t + y^{*2}_t); \qquad V = [(h/L)^2/12]\int_0^1 \theta_s^2 ds \tag{4}$$

The algorithm for the uncoiling motion is the following:

Choose an initial P and ϕ and find the static deformed shape.

Vary P and ϕ locally and find the change in each that maximizes the change in potential energy.

Take the new P and ϕ and find the new deformed shape.

Add the decrement in V to the kinetic energy T.

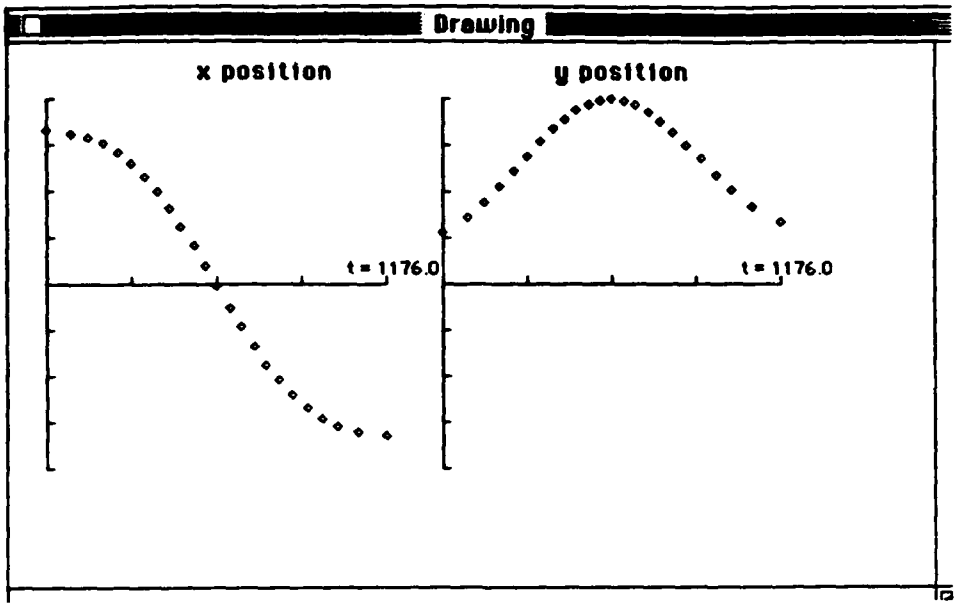
Use the change in T to find the velocity, and the change in position to find the time interval.

The rebending algorithm is the reverse of this, with a transfer of energy from the kinetic to the potential. It should be clear that a mechanism for damping is easily introduced. Figure 21 shows one half cycle of the motion of the end point of the beam shown in Figure 20. The upper half of the figure shows the motion of the end as a function of time. The lower half shows the actual shape of the beam at each time step. The mass ratio is 10 and $h/L = 0.01$. The time is measured in scaled time units, so that one can see that the motion is slow compared to the elastic wave propagation times, consistent with the quasistatic analysis.

11.3. Where Do We Go From Here?

Our first niggling task is to clean up the algorithm described in the previous section. Once this is done there are at least two simple tests of the algorithm: Does it reproduce the linear oscillation of a mass on a beam at small amplitudes? Can it reproduce experimental results?

The latter test leads naturally to the construction of an experimental facility, useful as well for the more general control problems outlined in Section 11.1. This



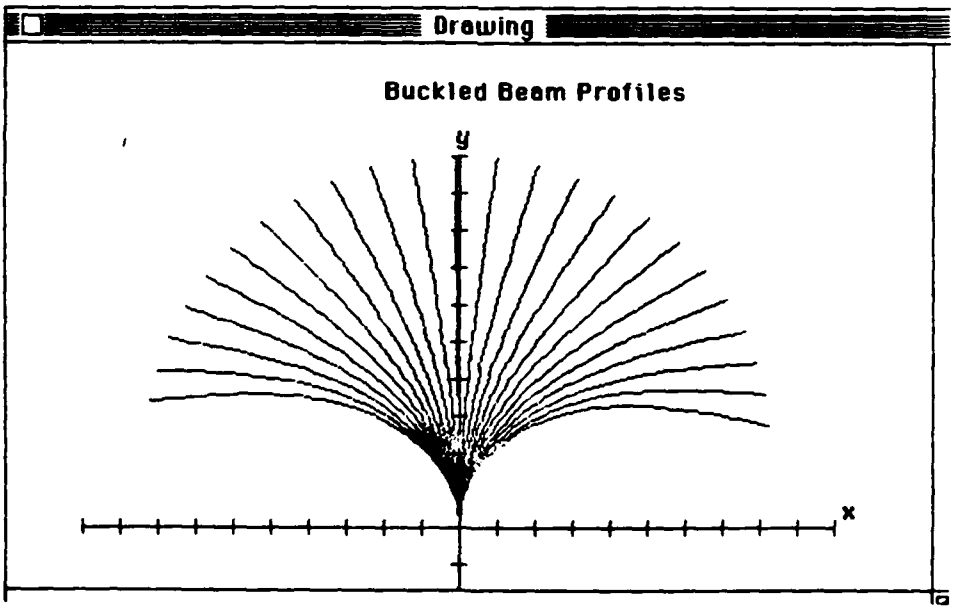


Figure 21

requires the design and construction of a small beam, like that at Oxford [Daniel et al. 1988], but considerably more flexible. A very thin plastic ruler gives the flavor of what we intend. Something of this size can be precisely driven using Compumotor drivers, allowing complex control algorithms to be tested. The behavior of such a system can be measured using strain gages on the beam and accelerometers on the mass. This effort would be conducted in parallel with improvements in the simulation programs. In particular, an alternative formulation of the dynamics suggests itself: construction of a set of Langrangian equations of motion from the

discretization of an integral expression of the Hamiltonian. This is similar, though not identical, to a finite element formulation of the problem.

In the more distant future we imagine scaling up to a beam large enough to allow the replacement of the mass by one or more cameras. Such a system would allow us to experiment with control algorithms relying on visual feedback. In parallel with this effort we would be able to begin work on a fully 3-D system, the real fishing pole that provided the motivation for the simple problems described in this document.